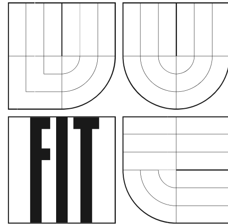
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Simulátor zásobníkového automatu

Bakalárska práca

2007 Martin Graizely

Simulátor zásobníkového automatu

Odovzdané na Fakultě informačních technologií Vysokého učení technického v Brně
dňa 24. januára 2007

© Martin Graizely, 2007

Táto práca vznikla ako školské dielo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práca je chránená autorským zákonom a jej použitie bez udelenia oprávnenia autorom je nezákonné, s výnimkou zákonom definovaných prípadov.

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením
Ing. Radka Bidla. Uviedol som všetky literárne pramene a publikácie,
z ktorých som čerpal.

. . . .
Martin Graizely
24. januára 2007

Abstrakt

Vizualizácia často napomáha pochopeniu problému a umožňuje jeho rýchle osvojenie. Táto práca je zameraná na implementáciu simulátora zásobníkového automatu. Účelom tejto aplikácie je uľahčiť pochopenie tohto abstraktného stroja a jeho možností, poskytnutím vizuálneho návrhu a umožnením interaktívnej animácie algoritmu.

Kľúčové slová

algoritmus, animácia, demonštrácia, determinizmus, diagram, editor, graf, gramatika, jazyk, prechodová funkcia, reťazec, simulácia, slovo, stav, symbol, syntax, zásobník, zásobníkový automat

Poďakovanie

Týmto by som chcel predovšetkým poďakovať pánovi Ing. Radkovi Bidlovi, vedúcemu mojej bakalárskej práce, za jeho cenné rady, pripomienky a veľkú ochotu pomôcť, ktorú prejavil počas tvorby tejto práce. Poďakovanie patrí aj mojej rodine a priateľom za ich veľkú toleranciu a podporu.

Abstract

Visualization often helps with the understanding and allows for a quicker grasp of the problem. This work is aimed on implementation of pushdown automaton simulator. The purpose of the application is to ease understanding of such abstract machine and its abilities, by providing visual design and possibility of algorithm's interactive animation.

Keywords

algorithm, animation, demonstration, determinism, diagram, editor, graph, grammar, language, transition function, string, simulation, word, state, symbol, syntax, pushdown, stack, pushdown automaton

Obsah

Obsah 6

1 Úvod 8

2 Úvod do teórie formálnych jazykov 9
- 2.1 Abeceda a reťazec 9
- 2.2 Operácie nad reťazcami 10
- 2.3 Jazyk 11
  - 2.3.3 Operácie nad jazykmi 11
- 2.4 Regulárne výrazy a regulárne jazyky 12
- 2.5 Konečný automat 12
  - 2.5.7 Deterministický konečný automat 14
  - 2.5.8 Pumping lemma 15
- 2.6 Zásobníkový automat 16

3 Rozbor problému 19
- 3.1 Návrh rozhrania 19
- 3.2 Návrh programu 20

4 Popis implementácie 21
- 4.1 Platforma .NET 21
  - 4.1.1 Common Language Runtime 22
  - 4.1.2 Common Language Infrastructure 22
  - 4.1.3 Common Type System 22
  - 4.1.4 Common Intermediate Language 22
  - 4.1.5 Knižnica tried .NET 23
- 4.2 Implementácia 23
  - 4.2.1 Kreslené prvky pracovnej plochy 23
  - 4.2.2 Logické prvky 25
  - 4.2.3 Editačné príkazy 25
  - 4.2.4 Komponenty 26
  - 4.2.5 Simulátor 27
  - 4.2.6 Nástroje 28
  - 4.2.7 Nastavenia programu 28

5 Práca so simulátorom 29
- 5.1 Editácia automatu 29
- 5.2 Simulácia 31

6 Záver **34**

Kapitola 1

Úvod

Prirodzené jazyky, ako ich poznáme, slúžia ľuďom na výmenu informácii a všetky účely komunikácie. Koncom 19. storočia začali vznikať umelé jazyky, ktoré slúžili na opis faktov a myšlienkových uzáverov. Sprvu to boli logické jazyky, ktoré umožňovali lepšie, prehľadnejšie a exaktnejšie vyjadrenie logických súvislostí. Popri logických počtoch vznikali s vývojom výpočtových zariadení od roku 1940 programovacie jazyky, ktoré sa museli taktiež presne definovať. Jazyk je vyjadrený systémom symbolov (tzv. *lexémy*) a *gramatikou*, čiže pravidlami, podľa ktorých z týchto symbolov tvoria vety jazyka. Na rozdiel od prirodzených jazykov, v ktorých sa slová, pravidlá a význam priebežne menia, umelé jazyky majú pevný základný slovník a pevnú *syntax* a sémantiku.

Štúdiu prirodzených jazykov sa venoval americký matematik Noam Chomsky, ktorý sa snažil o formuláciu všeobecnej teórie syntaxe prirodzených jazykov. V roku 1956 položil základy novej vednej disciplíny: Teórie formálnych jazykov. Chomsky skúmal rôzne typy formálnych jazykov a to, či je nimi možné opísať kľúčové vlastnosti ľudského jazyka. Vytvoril tak veľmi presné matematické modely gramatík a Chomskeho hierarchickú klasifikáciu formálnych jazykov. Táto hierarchia delí formálne gramatiky do štyroch tried s rastúcou vyjadrovacou silou tak, že každá nasledovná trieda je schopná generovať širšiu množinu formálnych jazykov ako predchádzajúca. Podľa Chomskeho ale modelovanie niektorých aspektov ľudského jazyka vyžaduje komplexnejšiu formálnu gramatiku ako tie, ktoré popisuje táto hierarchia.

V súčasnosti našli formálne jazyky veľmi široké uplatnenie hlavne v oblasti hardware a software (počítačové siete, kompresia dát). Ich najväčšie uplatnenie je v oblasti programovacích jazykov, hlavne v konštrukcii prekladačov a teórii *automatov*.

Pojmom automaty sa označujú najrôznejšie zariadenia spracovávajúce informácie, v závislosti od ktorých produkujú určitú formu výstupu. Automatom je napríklad komplikovaný radič zbernice na základnej doske počítača, ale aj jednoduchý elektrický obvod so žiarovkou a spínačom, ktorý má len dva stavy. Medzi týmito sa pohybuje na základe prepnutia spínača. V informatike sa pojmom automaty označujú abstraktné matematické modely strojov. V súvislosti s formálnymi jazykmi sa automaty používajú ako formálne systémy na opis jazykov, resp. na opis systémov spracúvajúcich jazyky.

V začiatku práce si predstavíme najzákladnejšie pojmy formálnych jazykov a vzťahy medzi nimi až k *zásobníkovému automatu*. Následne rozoberieme problém požiadaviek na vytváranú aplikáciu. Stručne bude predstavená vývojová platforma .NET a potom sa zameriame na opis samotnej implementácie simulátoru a jeho ovládania. Historické fakty v tomto texte boli čerpané z [8], [6], [1] a [7]

Kapitola 2

Úvod do teórie formálnych jazykov

Každý jazyk je definovaný postupnosťou znakov, ktoré sú zreťazené podľa určitých pravidiel. Formálnym jazykom sa označuje množina konečných *reťazcov* (tj. slov konečnej dĺžky) nad určitou *abecedou*. Takáto formálna výstavba viet a slov, ktoré patria k jazyku sa nazýva syntax. Syntax programovacieho jazyka musí byť jednoznačne definovaná aby bolo možné určiť ktoré znakové postupnosti sú korektne formulovanými programami jazyka a ktoré nie. Na opis syntaxe existuje celý rad metód:

- Možno definovať gramatiku, ktorá generuje jazyk zodpovedajúci práve množine všetkých korektných znakov. Takáto gramatika sa presnejšie nazýva *generatívnou gramatikou*.
- Možno uviesť automat, ktorý akceptuje práve syntakticky korektné postupnosti znakov.
- Pomocou syntaktických diagramov sa dá syntax opísať graficky.

Aby sme mohli tieto metódy predstaviť musíme sa najskôr oboznámiť s niektorými základnými pojmami z teórie formálnych jazykov. Pre jednotnosť výkladu použijeme formálne definície podľa [5].

2.1 Abeceda a reťazec

Definícia 2.1.1 *Abeceda* Σ je konečná neprázdna množina znakov nazývaných symboly.

Postupnosť symbolov tvorí reťazec. *Prázdny reťazec* značený ako ε neobsahuje žiadne symboly. Nasledujúca definícia definuje rekurzívne slovo nad abecedou.

Definícia 2.1.2 Nech Σ je abeceda.

1. ε je reťazec nad Σ.
2. Ak x je reťazec nad Σ a $a \in \Sigma$, potom xa je reťazec nad Σ.

Dĺžka reťazca x je počet všetkých symbolov v reťazci x.

Definícia 2.1.3 Nech x je reťazec nad abecedou Σ. *Dĺžka reťazca* x, $|x|$, je definovaná následovne:

1. ak $x = \varepsilon$, potom $|x| = 0$
2. ak $x = a_1 \dots a_n$ pre nejaké $n \geq 1$, kde $a_i \in \Sigma$ pre všteky $i = 1, \dots, n$, potom $|x| = n$.

2.2 Operácie nad reťazcami

Nad reťazcami sú definované nasledujúce operácie, ktoré nám o nich umožňujú získavať informácie, alebo s nimi manipulovať.

Definícia 2.2.1 Nech x a y sú reťazce nad abecedou, Σ. Potom xy je *zreťazením* x a y.

Platí že

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

Definícia 2.2.2 Nech x je reťazec nad abecedou, Σ. Pre $i \geq 0$, je i-tá *mocnina* x rekurzívne definovaná následovne

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

Definícia 2.2.3 Nech x je reťazec nad abecedou, Σ. *Reverzia* reťazca x, reversal(x) je definovaná následovne

1. ak $x = \varepsilon$, potom reversal(x)$= \varepsilon$
2. ak $x = a_1 \dots a_n$, pre nejaké $n \geq 1$, a $a_i \in \Sigma$, pre $i = 1 \dots, n$, potom reversal($a_1 \dots a_n$) = $a_n \dots a_1$.

Definícia 2.2.4 Nech x a y sú reťazce nad abecedou, Σ. Potom, x je *prefix* y, ak existuje reťazec, z, nad abecedou Σ, taký že $xz = y$; naviac platí, že ak $x \notin \{\varepsilon, y\}$, tak x je *vlastným prefixom* y.

Pre slovo y, označuje prefix(y) množinu všetkých prefixov y; tj.

$$\text{prefix}(y) = \{x : x \text{ je prefix } y\}$$

Definícia 2.2.5 Nech x a y sú reťazce nad abecedou, Σ. Potom, x je *sufix* y, ak existuje reťazec, z, nad abecedou Σ, taký že $zx = y$; naviac platí, že ak $x \notin \{\varepsilon, y\}$, tak x je *vlastným sufixom* y.

Pre slovo y, označuje sufix(y) množinu všetkých sufixov y; tj.

$$\text{sufix}(y) = \{x : x \text{ je sufix } y\}$$

Definícia 2.2.6 Nech x a y sú reťazce nad abecedou, Σ. Potom, x je *podreťazcom* y, ak existujú dva reťazce, z s z', nad abecedou Σ, také že $zxz' = y$; naviac platí, že ak $x \notin \{\varepsilon, y\}$, tak x je *vlastným podreťazcom* y.

Pre slovo y, označuje podreťazec(y) množinu všetkých podreťazcov y; tj.

$$\text{podreťazec}(y) = \{x : x \text{ je podreťazec } y\}$$

2.3 Jazyk

Uvažujme abecedu, Σ a jej *iteráciu* Σ^*, ktorá značí množinu všetkých reťazcov nad Σ, teda aj prázdneho reťazca. *Pozitívna iterácia* množiny Σ, $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ obsahuje už len všetky neprázdne reťazce nad Σ. Jazyk nad abecedou Σ je definovaný ako podmnožina Σ^*.

Definícia 2.3.1 Nech Σ je abeceda, a nech $L \subseteq \Sigma^*$. Potom L je *jazyk* nad abecedou Σ.

Jazyk nazývame konečným resp. nekonečným podľa toho, či obsahuje konečný resp. nekonečný počet reťazcov. Pripomeňme, že pre množinu S vyjadruje card(S) počet jej prvkov.

Definícia 2.3.2 Nech L je ľubovoľný jazyk. L je *konečný* ak card(L)$= n$, pre nejaké $n \geq 0$; v opačnom prípade je L *nekonečný*.

2.3.3 Operácie nad jazykmi

Pretože sú jazyky množiny reťazcov, sú nad nimi definované rovnaké operácie ako pre množiny, tj. zjednotenie, prienik, rozdiel a doplnok. Uvažujme dva jazyky L_1 a L_2. Výsledkom týchto operácií je opäť jazyk.

$$L_1 \cup L_2 = \{x : x \in L_1 \vee x \in L_2\}$$

$$L_1 \cap L_2 = \{x : x \in L_1 \wedge x \in L_2\}$$

$$L_1 - L_2 = \{x : x \in L_1 \wedge x \notin L_2\}$$

$$\bar{L} = \Sigma^* - L$$

Ďalej nad nimi možno definovať tie isté operácie, aké poznáme pre reťazce.

Definícia 2.3.4 Nech L_1 a L_2 sú dva jazyky. *Zreťazenie* L_1 a L_2, L_1L_2, je definované ako

$$L_1L_2 = \{xy : x \in L_1 \wedge y \in L_2\}$$

Definícia 2.3.5 Nech L je jazyk. *Reverzia* L, reverzia(L), je definovaná ako

$$reverzia(L) = \{reverzia(x) : x \in L\}$$

Definícia 2.3.6 Nech L je jazyk. Pre $i \geq 0$, i-ta *mocnina* L, L^i, je definovaná ako

- $L^0 = \varepsilon$
- pre všetky $i \geq 1$, $L^i = LL^{i-1}$.

Definícia 2.3.7 Nech L je jazyk. *Iterácia* L, L^*, je definovaná ako

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

Definícia 2.3.8 Nech L je jazyk. *Pozitívna iterácia* L, L^+, je definovaná ako

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

2.4 Regulárne výrazy a regulárne jazyky

Pod pojmom *regulárne výrazy* rozumieme znakové reťazce, ktoré nám umožňujú deklaratívnou formou popísať určitú množinu reťazcov. Stretávame sa s nimi napríklad pri použití vyhľadávacích príkazov, validátoroch vstupov vo formulároch alebo generátoroch lexikálnych analyzátorov. Je nimi možné opísať väčšinu lexémov programovacieho jazyka ako *regulárne jazyky* [5]. Regulárne výrazy definujú regulárne jazyky pomocou operácií zreťazenia, zjednotenia a iterácie.

Definícia 2.4.1 Nech Σ je abeceda. *Regulárny výraz* nad Σ a jazyky, ktoré tento výraz opisuje, sú definované rekurzívne následovne:

1. $\varnothing$ je regulárny výraz vyjadrujúci prázdnu množinu.
2. ε je regulárny výraz vyjadrujúci $\{\varepsilon\}$.
3. ak r a s sú regulárne výrazy vyjadrujúce jazyky R a S, potom
   (a) $(r \cdot s)$ je regulárny výraz vyjadrujúci RS
   (b) $(r+s)$ je regulárny výraz vyjadrujúci $R \cup S$
   (c) (r^*) je regulárny výraz vyjadrujúci R^*.

Definícia 2.4.2 Nech L je jazyk nad abecedou Σ. L je *regulárny jazyk* nad Σ ak $L = L(r)$ pre regulárny výraz r nad Σ.

2.5 Konečný automat

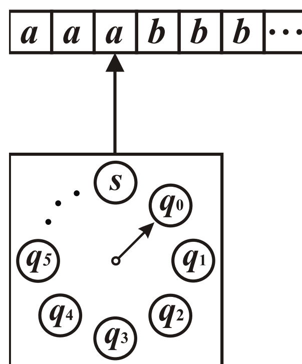
Ako už bolo spomenuté v úvode, automaty sú zariadenia, ktoré na základe vstupných informácií produkujú určitý výstup. Existuje množstvo systémov, o ktorých môžme tvrdiť, že sa v každom časovom okamihu nachádzajú v istom stave, ktorý je jedným z konečného počtu stavov. Účelom týchto stavov je zapamätanie relevantného množstva informácií o minulosti systému. *Konečný automat* možno znázorniť ako riadiaci mechanizmus disponujúci konečnou množinou stavov a vstupnou a výstupnou páskou. Špeciálnym prípadom sú rozpoznávacie konečné automaty, ktoré nemajú výstupnú pásku, tzv. *konečné akceptory*, o ktorých budeme ďalej pojednávať. Budeme pri tom aj naďalej používať všeobecný pojem konečný automat. Tento mechanizmus je zjednodušene znázornený na obrázku 2.1. Automat pracuje tak, že na začiatku činnosti sa riadiaci mechanizmus nachádza v počiatočnom stave a čítacia hlava stojí na prvom znaku vstupnej pásky. Automat prečíta prvý symbol a zvolí podľa neho nasledujúci stav. Pri tom môže posunúť hlavu o jednu pozíciu doprava. Čítacia hlava sa po páske môže pohybovať iba jedným smerom. Hovoríme, že konečný automat vstup *akceptuje*, ak začína prijímať symbol z pásky v počiatočnom stave a po prečítaní posledného symbolu pásky sa nachádza v koncovom stave [1]. V texte sa stručne zameriame na použitie niektorých konečných automatov slúžiacich k opisu systémov spracovávajúcich jazyky.

Definícia 2.5.1 *Konečný automat* je pätica:

$$M = (Q, \Sigma, R, s, F)$$

kde

Q je konečná neprázdna množina stavov



Obr. 2.1: Konečný automat

Σ je vstupná abeceda taká, že $\Sigma \cap Q = \varnothing$

$R \subseteq Q(\Sigma \cup \{\varepsilon\}) \times Q$ je relácia

$s \subseteq Q$ je počiatočný stav

$F \subseteq Q$ je mnnožina koncových stavov.

R vyjadruje konečnú množinu pravidiel. Pravidlo, $(pa, q) \in R$, kde $p, q \in Q$ a $a \in \Sigma \cup \{\varepsilon\}$. Zapisujeme aj ako

$$pa \rightarrow q$$

Toto pravidlo popisuje zmenu stavu q na stav p, prečítaním znaku a zo vstupnej pásky. Pravidlo v tvare $r : p \rightarrow q \in R$, znamená že hlava sa na páske neposunie. Nazývame ho ε-*pravidlo*.

Definícia 2.5.2 *Konfigurácia* automatu $M = (Q, \Sigma, R, s, F)$, je reťazec χ vyhovujúci predpisu

$$\chi = Q\Sigma^*$$

Konfigurácia vyjadruje okamžitú situáciu automatu špecifikovaním aktuálneho stavu, v ktorom sa automat nachádza a doposiaľ neprečítaným vstupom na páske.

Definícia 2.5.3 Nech $M = (Q, \Sigma, R, s, F)$ je konečný automat. Pravú stranu pravidla r, takého že $r \in R$, označme ako rhs(r) a ľavú ako lhs(r). Potom ak lhs(r)y je konfiguráciou M, kde $y \in \Sigma^*$, tak M vykonáva *krok výpočtu* z lhs(r)y do rhs(r)y podľa pravidla r. Značíme:

$$\text{lhs}(r)y \vdash \text{rhs}(r)y\ [r]$$

Pokiaľ nie je potrebné upresnenie použitého pravidla v zápise $\text{lhs}(r)y \vdash \text{rhs}(r)y\ [r]$, možno použiť skrátený zápis

$$\text{lhs}(r)y \vdash \text{rhs}(r)y$$

Definícia 2.5.4 Nech $M = (Q, \Sigma, R, s, F)$ je konečný automat, a nech χ_0 až χ_n sú konfigurácie M. *Postupnosť'krokov*, $\chi_0 \vdash^n \chi_n$ označuje sled n prechodov medzi konfiguráciami automatu:

$$\chi \vdash^0 \chi\ [\varepsilon]$$
$$\chi_0 \vdash^n \chi_n\ [r_1 \dots r_n]$$

Na základe tejto definície môžno vyjadriť pojmy $\vdash^+$ a $\vdash^*$. Prvý vyjadruje tranzitívny uzáver $\vdash$, a druhý tranzitívny a reflexívny uzáver $\vdash$.

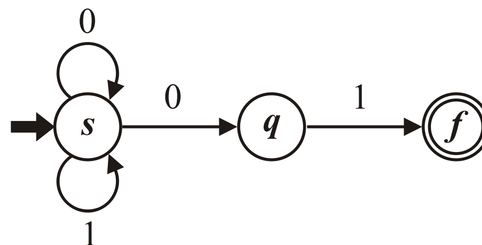
Definícia 2.5.5 Nech χ a χ' sú dve konfigurácie konečného automatu M.

1. Ak existuje $n \geq 1$ také, že $\chi \vdash^n \chi'$ v M, potom $\chi \vdash^+ \chi'$.
2. Ak existuje $n \geq 0$ také, že $\chi \vdash^n \chi'$ v M, potom $\chi \vdash^* \chi'$.

Konečné automaty je možné reprezentovať graficky ako orientované grafy tzv. *prechodovými diagramami*:

1. Stavy tvoria uzly grafu a sú reprezentované kružnicami.
2. Zo stavu vedie hrana označená vstupným symbolom do ďalšieho stavu, ak existuje takéto pravidlo v R.
3. Počiatočný stav sa znázorňuje so vstupujúcou šípkou, ktorej začiatok nie je v žiadnom stave.
4. Koncové stavy sa značia dvojitou kružnicou.

Príklad 1 Uveďme na obrázku 2.2 príklad grafického popisu konečného automatu $M = (Q, \Sigma, R, s, F)$ definovaného ako $Q = \{s, q, f\}$, $\Sigma = \{0, 1\}$, $R = \{s0 \to s, s1 \to s, s0 \to q, q1 \to f\}$, $F = \{f\}$.



Obr. 2.2: Prechodový diagram

Z uvedených definícií je zrejmé, že konečné automaty dokážu prijímať určité množiny slov, teda určité jazyky. V tomto momente môžme uviesť definíciu jazyka, ktorý je *prijímaný konečným automatom M*.

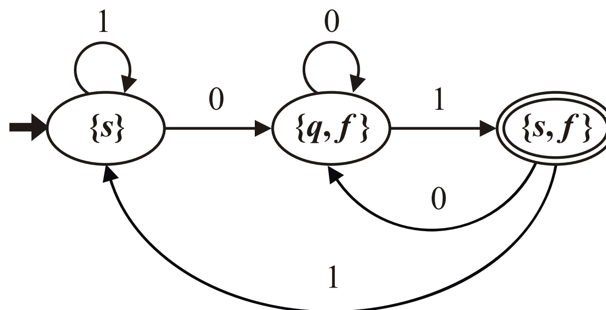
Definícia 2.5.6 Jazyk L *prijímaný konečným automatom* $M = (Q, \Sigma, R, s, F)$ je definovaný ako

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

2.5.7 Deterministický konečný automat

Z obrázku 2.2 je vidieť, že automat môže prejsť čítaním symbolu 0 do stavov s a q. Takéto automaty, ktoré majú z určitého stavu na výber viac ako jedno pravidlo pre rovnaký symbol, sa nazývajú *nedeterministické konečné automaty* (NKA). *Deterministickým konečným automatom* (DKA) označujeme konečný automat, ktorý spĺňa podmienku takú, že po prečítaní konkrétneho symbolu zo vstupu je v každom stave možné jednoznačne určiť najviac jeden nasledujúci stav. Dá sa dokázať, že ku každému nedeterministickému konečnému automatu A je možné skonštruovať ekvivalentný deterministický konečný automat B, ktorý rozpoznáva ten istý jazyk, t.j. $L(A) = L(B)$. Spomeňme len, že existuje niekoľko druhov navzájom ekvivalentných variant konečných automatov odlišných

špecifikáciou ale s rovnakou silou. Ich vzájomnými prevodmi sa nebudeme detailne zaoberať. *Sila modelu* automatu je množina jazykov, ktoré daný model automatu dokáže popísať. Na obrázku 2.3 je uvedený DKA skonštruovaný z vyššie uvedeného NKA. Stavy tu reprezentujú prvky z potenčnej množiny Q nedeterministického automatu M, takže každý stav takto vytvoreného automatu odpovedá istej množine stavov pôvodného nedeterministického automatu a sú medzi nimi jednoznačné prechody.



Obr. 2.3: Deterministický konečný automat

2.5.8 Pumping lemma

Pumping lemma (lema o vkladaní) je tvrdenie vyjadrujúce vlastnosť, ktorú musia reg. jazyky spĺňať. To znamená, že je podmienkou nutnou, ale nie postačujúcou k tomu aby jazyk patril do triedy regulárnych jazykov. Možno ju teda využiť na dôkaz sporom, že daný jazyk nie je regulárny. Myšlienka spočíva vo fakte že ak je dĺžka akceptovaného reťazca väčšia ako počet stavov automatu, znamená to, že sa automat nachádzal v nejakom stave viac ako raz a jeho diagram teda obsahuje slučku. Z tohto vyplýva možnosť opakovania určitých podreťazcov v reťazcoch jazyka opísaného týmto automatom. To znamená, že vyňatím opakujúcej sa sekvencie z reťazca, bude tento reťazec stále prijateľný automatom a bude vyhovovať definícii jazyka.

Definícia 2.5.9 Nech L je regulárny jazyk. Potom existuje prirodzené číslo k také že každý reťazec, $z \in L$ splňujúci podmienku $|z| \geq k$ môže byť vyjadrený ako

$$z = uvw$$

kde

1. $v \neq \varepsilon$
2. $|uv| \leq k$
3. $uv^m w \in L$, pre všetky $m \geq 0$.

Príklad 2 Asi najznámejším príkladom demonštrujúcim jazyk, ktorý nespĺňa pumping lemma je jazyk

$$L = \{a^n b^n : n \geq 0\}$$

Ukážeme si na ňom dôkaz podľa [5]. Postupujeme sporom, teda predpokladáme že jazyk L je regulárny. Podľa lemmy existuje konštanta $k \geq 0$. Uvažujeme reťazec

$$z = \{a^k b^k\}$$

Keďže $|z| = 2k$ a $2k > k$, takže platí $|z| \geq k$. Podľa lemmy môžme z rozdeliť na uvw tak, aby platili jej tri podmienky. Keďže $|v| \geq 1$,

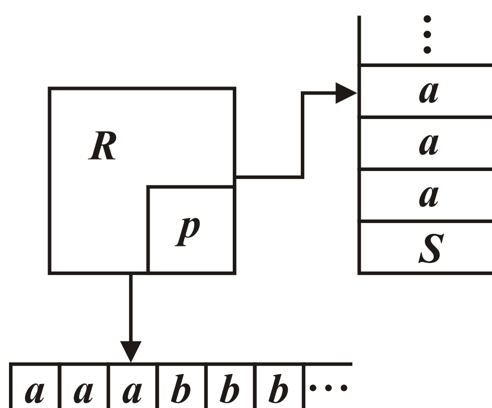
$$v \in \{a\}^+ \cup \{b\}^+ \cup \{a\}^+\{b\}^+$$

1. Predpokladajme $v \in \{a\}^+$. Uvažujeme $uv^0w = uw$. Potom, $uw = a^jb^k$, pre nejaké $j \leq k-1$, takže $uw \notin L$. To je spor s treťou podmienkou lemmy.
2. Predpokladajme $v \in \{b\}^+$. Uvažujeme $uv^0w = uw$. Potom, $uw = a^kb^j$, pre nejaké $j \leq k-1$, takže $uw \notin L$ a to je spor.
3. Predpokladajme $v \in \{a\}^+\{b\}^+$. Uvažujeme $uv^2w = uvvw$. Pretože $v \in \{a\}^+\{b\}^+$, $uvvw \in \{a\}^*\{a\}^+\{b\}^+\{a\}^+\{b\}^+\{b\}^*$, takže $uvvw \notin L$ a to je spor.

Pre každý z troch rozkladov existuje $m \geq 0$ také že $uv^mw \notin L$ a všetky teda vedú k sporu. Preto L nie je regulárny.
Tento dôkaz znamená, že konečnými automatmi nemôžeme analyzovať syntax programovacích jazykov, ktoré môžu mať ľubovoľne veľa zanorených štruktúr tvoriacich výrazy a bloky kódu. Konečný automat si teda nedokáže zapamätať koľko prečítal otváracích zátvoriek aby ich tak mohol spárovať so zátvorkami uzatváracími.

2.6 Zásobníkový automat

V tejto kapitole bude predstavený zásobníkový automat. Ako bolo ukázané pri lemme o vkladaní, konečné automaty nemajú možnosť prijímať jazyky, u ktorých je podstatné zapamätanie nejakej dodatočnej informácie. Tento problém sa dá jednoducho a veľmi účinne vyriešiť dodaním pamäte, s ktorou by vedel automat disponovať. Zásobníkový automat ako už názov napovedá, využíva pre tento účel pamäť LIFO čiže zásobník, viz obrázok 2.4. Pomocou zásobníku si automat môže pamätať ľubovoľne veľa znakov a uskutočňovať nasledujúce stavové prechody v závislosti od naposledy zapamätaného znaku. Pri tom sa symbol na vrchole zásobníka nahradí reťazcom.



Obr. 2.4: Zásobníkový automat

Definícia 2.6.1 *Zásobníkový automat* je pätica

$$M = (Q, \Sigma, R, s, F)$$

kde

Q je konečná množina stavov;

Σ je abeceda taká že $\Sigma \cap Q = \varnothing$ a $\Sigma = \Sigma_{\text{I}} \cup \Sigma_{\text{PD}}$, kde Σ_{I} je *vstupnou abecedou*, a Σ_{PD} je *abecedou zásobníku* obsahujúcou *počiatočný symbol S*;

$R \subseteq \Sigma_{\text{PD}}Q(\Sigma_{\text{I}} \cup \{\varepsilon\}) \times \Sigma_{\text{PD}}^{*}Q$ je konečná relácia;

$s \in Q$ je *počiatočný stav*;

$F \in Q$ je množina *koncových stavov*.

Členy R sa nazývajú pravidlami, a preto sa R označuje aj ako konečná množina pravidiel. Uvažujme pravidlo $(Apa, wq) \in R$, kde $A \in \Sigma_{\text{PD}}, p, q \in Q, a \in \Sigma_{\text{I}} \cup \{\varepsilon\}$, a $w \in \Sigma_{\text{PD}}^{*}$. Namiesto (Apa, wq) budeme používať zápis v tvare:

$$Apa \to wq$$

Definícia 2.6.2 Nech $M = (Q, \Sigma, R, s, F)$ je zásobníkový automat. *Konfiguráciou M* je reťazec χ pre ktorý platí

$$\chi \in \Sigma_{\text{PD}}^{*}Q\Sigma_{\text{I}}^{*}$$

Definícia 2.6.3 Nech $M = (Q, \Sigma, R, s, F)$ je zásobníkový automat. Ak je konfiguráciou M, kde $x \in \Sigma_{\text{PD}}^{*}, y \in \Sigma_{\text{I}}^{*}$, a $r \in R$, potom M vykonáva *krok výpočtu z* $x\text{lhs}(r)y$ *do* $x\text{rhs}(r)y$ *podľa* r nasledovne

$$x\text{lhs}(r)y \vdash x\text{rhs}(r)y \ [r]$$

Pokiaľ nie je potrebné upresnenie použitého pravidla v predchádzajúcom zápise, možno použiť skrátený zápis

$$x\text{lhs}(r)y \vdash x\text{rhs}(r)y$$

Definícia 2.6.4 Nech $M = (Q, \Sigma, R, s, F)$ je zásobníkový automat.

1. Nech χ je nejaká konfigurácia M. M vykoná nula krokov podľa ε, zapisané ako

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

2. Nech χ_0 až χ_n sú konfigurácie M, pre $n \geq 1$ také že

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

kde $r \in R, i = 1, \ldots, n$; také že

$$\begin{aligned} \chi_0 &\vdash \chi_1 \ [r_1] \\ &\vdash \chi_2 \ [r_2] \\ &\vdots \\ &\vdash \chi_n \ [r_n] \end{aligned}$$

Potom M *vykonáva n krokov z* χ_0 *do* χ_n *podľa* $r_1 \dots r_n$, zapísaných ako

$$\chi_0 \vdash^n \chi_n \; [r_1 \dots r_n]$$

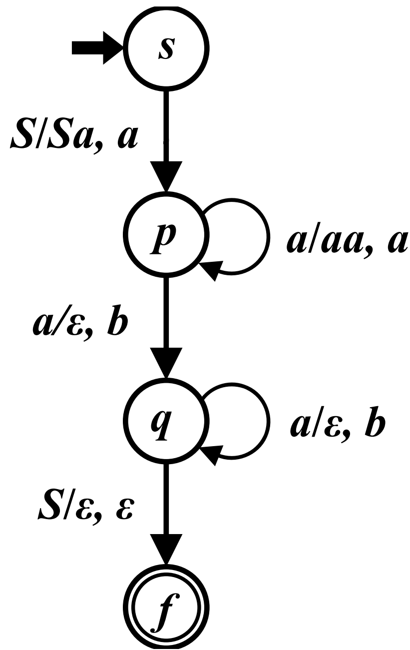
Na základe $\vdash^n$ môžno vyjadriť pojmy $\vdash^+$ a $\vdash^*$. Prvý vyjadruje tranzitívny uzáver $\vdash$, a druhý tranzitívny a reflexívny uzáver $\vdash$.

Definícia 2.6.5 Nech χ a χ' sú dve konfigurácie zásobníkového automatu M.

1. Ak existuje $n \geq 1$ také, že $\chi \vdash^n \chi'$ v M, potom $\chi \vdash^+ \chi'$.
2. Ak existuje $n \geq 0$ také, že $\chi \vdash^n \chi'$ v M, potom $\chi \vdash^* \chi'$.

Definícia 2.6.6 Nech $M = (Q, \Sigma, R, s, F)$ je zásobníkový automat a $w \in \Sigma_{\mathrm{I}}^*$. Ak existuje prijímajúci výpočet v tvare $Ssw \vdash^* zf$ v M, kde $z \in \Sigma_{\mathrm{PD}}^*$ a $f \in F$, potom M *prijíma* w. *Jazyk akceptovaný* M, $L(M)$, je definovaný ako

$$L(M) = \{w : w \in \Sigma_{\mathrm{I}}^* \text{ a } Ssw \vdash^* zf \text{ v } M \text{pre nejaké } z \in \Sigma_{\mathrm{PD}}^* \text{ a } f \in F\}$$



Obr. 2.5: Grafická reprezentácia zásobníkového automatu

Kapitola 3

Rozbor problému

3.1 Návrh rozhrania

Riešená práca spočíva v implementácii grafického editoru zásobníkového automatu s možnosťou interaktívnej simulácie prijímania reťazca. Účelom je poskytnúť demonštračný prostriedok, ktorý by mohol byť využitý ako pomôcka pri štúdiu formálnych jazykov. V prvej fáze návrhu bolo nutné vyjadriť požiadavky na funkcionalitu z hľadiska užívateľského rozhrania a funkcií simulátora. Tie boli formulované nasledovne.

Užívateľské rozhranie

- Rozhranie musí obsahovať celú vizuálnu reprezentáciu automatu, čo okrem editačnej plochy znamená i zásobník, vstupnú pásku a prehľadný zoznam pravidiel korešpondujúci s vizuálnou definíciou zásobníkového automatu.
- Ovládanie by malo byť praktické a jednoduché, čiže užívateľsky priateľské.
- Program má užívateľa prehľadne a vecne informovať o vykonávaných úlohách a prípadných problémoch.

Editácia

- Automat musí byť zostrojiteľný výlučne pomocou jeho grafickej notácie, tj. prechodovým diagramom.
- Notácia a celkový vzhľad by mali v najvyššej možnej miere korešpondovať s grafickou prezentáciou používanou v prednáškach predmetu Formálne jazyky a prekladače, prípadne takéto nastavenie umožňovať.
- Editačná plocha má byť neobmedzená veľkosťou okna aplikácie.

Simulácia

- Animované simulovanie práce automatu bude umožnené ovládacími prvkami na krok výpočtu, spustenie, pozastavenie a zastavenie simulácie.
- K dispozícii bude aj možnosť skoku v simulácii, aby užívateľ nebol animáciou po každé nútený čakať.

- Nastavenie časového intervalu pre dobu kroku simulácie.
- Užívateľovi bude v prípade nedeterminizmu ponúknutý výber relácie prechodu.
- Voľba metódy prijatia vstupného reťazca.
- Stavové zmeny budú zvýraznené tak, aby bolo možné sledovať ich súvislosti.

Ostatné požiadavky

- K dispozícii bude možnosť testu na determinizmus vytvoreného modelu.
- Nájdené nedeterministické prechody budú po vykonaní testu vyznačené a vypísané na výstupe.
- Modely bude možné ukladať do súborov.

3.2 Návrh programu

Dobrý návrh programu sa ukázal ako nutná podmienka pre jeho úspešnú implementáciu. Svoje uplatnenie v ňom našlo niekoľko návrhových vzorov a myšlienkových konceptov, ktoré z nich čerpali inšpiráciu aspoň čiastočne. Použité boli napríklad vzory Observer, Mediator, Command, Template, Visitor alebo Singleton. Hlavný problém predstavovalo oddelenie grafickej časti automatu na strane užívateľa a logickej časti ako jadra simulácie, keďže celá definícia automatu je vytváraná výlučne kreslenými prvkami jeho grafickej reprezentácie. Súčasne tu ale musia existovať určité väzby, ktoré zaistia budovanie spoločného systému definície zásobníkového automatu ako vyobrazuje obrázok 3.1. Problém tak bol rozdelený na tri riešené vrstvy:

- užívateľské rozhranie,
- štruktúra definície opisujúcej automat,
- logika predstavujúca samotný mechanizmus zásobníkového automatu.

Definícia automatu je vytváraná grafickým rozhraním na jednej strane a zároveň využívaná logikou simulátoru na strane výkonnej. V prvej fáze bola navrhnutá a implementovaná grafická a definičná časť s ohľadom na niektoré budúce potreby logickej vrstvy. Boli tak zohľadnené hlavné požiadavky na interaktivitu logiky s animovanou simuláciou. Vytvorenie logickej vrstvy ako najmenšej z trojice v záverečnej fáze tak už nebolo zložité. Zaujímavou úlohou sa ukázala byť aj implementácia možností Späť a Znova alebo vstupnej pásky ako samočinného komponentu.

Obr. 3.1: Návrh programu

Kapitola 4

Popis implementácie

4.1 Platforma .NET

Program je implementovaný v jazyku C♯ 2.0 na platforme .NET verzii 2[1]. V dnešnej dobe je .NET doplnkovou súčasťou systémov Microsoft Windows 95 až XP a integrovanou súčasťou Windows Vista. Informácie v tejto kapitole boli čerpané z [4] a [3]. Táto platforma je programovým rozhraním pre Windows služby a API. Integruje veľké množstvo technológií, ktoré vznikali v Microsofte počas neskorých 90tych rokov. Sú v nej začlenené technológie COM+, vývojová platforma pre web ASP, integrovaná je väzba na XML, objektovo orientovaný dizajn a podpora pre protokoly webových služieb ako napr. SOAP, WSDL a UDDI. Pri jej vývoji boli zohľadnené skúsenosti softvérového priemyslu s vývojom širokej škály náročných podnikových a webových aplikácií, na ktoré sú dnes kladené požiadavky ako podpora vzájomnej spolupráce, škálovateľnosť, dostupnosť a spravovateľnosť. .NET podporuje vývoj novej generácie aplikácií a webových služieb s využitím otvorených internetových štandardov (HTML, XML, SOAP). Jej účelom je plnenie nasledujúcich cieľov:

1. Poskytovať konzistentné programovacie prostredie, či už sa jedná o kód uložený a vykonávaný lokálne, vykonávaný lokálne ale distribuovaný Internetom, alebo vykonávaný na diaľku (Objekty sú medzi serverom a klientom prenášané využitím vyššie uvedených štandardov).

2. Poskytovať prostredie pre vykonávanie kódu, ktoré minimalizuje nasadenie softvéru a verzovacie konflikty (problém "DLL Hell", registrácia komponentov, nastavenia registrov)

3. Poskytovať prostredie pre vykonávanie kódu podporujúce bezpečné vykonávanie kódu, vrátane kódu vytvoreného neznámymi alebo čiastočne dôveryhodnými spoločnosťami.

4. Poskytovať prostredie pre vykonávanie kódu, ktoré eliminuje výkonnostné problémy skriptovaných alebo interpretovaných prostredí.

5. Poskytovať programátorom jednotnú konzistentnú skúsenosť naprieč rozdielnymi typmi aplikácií, ako sú aplikácie pre Windows a webové aplikácie (Rovnaké programovanie webov aj okien).

6. Stavať všetku komunikáciu na priemyselných štandardoch, aby sa zabezpečila možnosť integrácie kódu založeného na .NET s akýmkoľvek iným kódom (Štandardizovaná jazyková infraštruktúra).

[1] Znak ♯ (*krížik*) pochádza z hudobnej notácie, kde dvíha notu o pol stupňa. V praxi sa ale častejšie používa znak pre číslo #, keďže krížik sa na klávesnici nevyskytuje.[2]

Nasledujúci text bližšie opisuje architektúru spoločných prvkov platformy.

4.1.1 Common Language Runtime

CLR je prostredie, ktoré vykonáva kód .NET aplikácie. Je to ekvivalent k Javovskému Virtuálnemu Stroju (JVM). Táto infraštruktúra abstrahuje rozdiely podloženej platformy, akým je napríklad závislosť na CPU. Prekladá a spravuje vykonávaný kód, zaobstaráva správu pamäte a vlákien, vzdialený prístup ako aj zabezpečenie typovej kontroly a iných foriem správnosti kódu. Takto orientovaný kód sa nazýva spravovaným (managed code). Kód, ktorý nie je cielený pre toto prostredie je tzv. nespravovaný (unmanaged).

4.1.2 Common Language Infrastructure

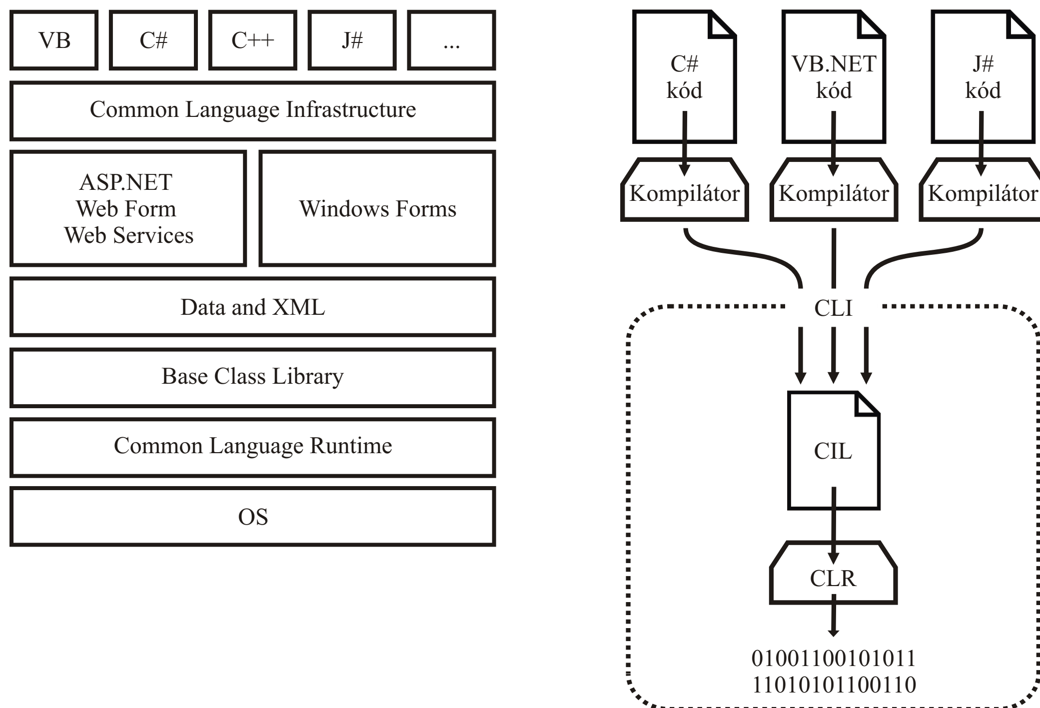
Spoločná jazyková infraštruktúra (CLI) je otvorená špecifikácia spoločnosti Microsoft, ktorá popisuje spustiteľný kód a prostredie pre beh programu tvoriace jadro platformy .NET. Táto špecifikácia je štandardizovaná organizáciami ISO a ECMA. Definuje prostredie umožňujúce viacerým vysokoúrovňovým jazykom byť použité na rôznych platformách (x86, Alpha, Itanium) bez nutnosti prepisovania kódu pre konkrétnu architektúru. V ponuke Microsoftu sú jazyky C♯, Visual Basic .NET, Visual J♯, Visual C++ a JScript. V ponuke tretích strán je viac ako 30 jazykov (COBOL, Pascal, Eiffel, Haskell, Perl, Python, Smalltalk, ...). Konkrétnou implementáciou tejto špecifikácie pre Windows je CLR. Open-source implementáciou CLI a sprievodných technológií sa zaoberá platforma Mono sponzorovaná firmou Novell.

4.1.3 Common Type System

Robustnosť kódu zabezpečuje CLR implementovaním striktnej typovej a kódovej verifikačnej infraštruktúry nazývanej Common Type System (CTS). CTS nešpecifikuje konkrétnu syntax. Jedná sa o všeobecnú koncepciu typov ako sú triedy, rozhrania a delegáti. Tieto typy môžu byť využité mnohými inými syntaxami jazykov. Väčšina jazykov používa pre tieto typy vlastné názvy ako aliasy pre tieto typy. Napríklad pre CTS triedu System.Int32 definuje C♯ alias s názvom int. Všetko v CTS je objekt a každý objekt je implicitne zdedený z jedinej základnej triedy, ktorá sa nazýva System.Object. Podobne hodnotové typy pochádzajú z triedy System.ValueType, ktorá je potomkom System.Object. S týmito typmi narába CLR ako s klasickými hodnotovými typmi a alokuje ich tak na zásobníku alebo inline v štruktúre. CTS zaisťuje, že každý spravovaný kód je samopopisujúci, čo v praxi znamená možnosť interoperability kódu napísaného v rôznych jazykoch. Takto je umožnené dedenie tried, zachytávanie výnimiek a využitie výhod polymorfizmu naprieč rôznymi jazykmi. Akýkoľvek jazyk, ktorý má byť kompatibilný s CLR môže definovať vlastnú syntax, ale musí využívať aspoň časť typov definovaných v CTS. Rozsah využitia možností CTS je na voľbe tvorcu jazyka.

4.1.4 Common Intermediate Language

Nazývaný aj Microsoft Intermediate Language (MSIL) je jediným pravým jazykom platformy. Jazyky cielené na .NET sa prekladajú do tohto medzi-kódu, ktorý je zostavený do byte kódu. CIL je platformovo nezávislá inštrukčná sada, ktorá môže byť vykonávaná v akomkoľvek prostredí podporujúcom platformu .NET. Jazyk je dobre čitateľný a je možné v ňom programovať. Kód nie je interpretovaný, ale do konečnej podoby strojového kódu ho za behu prekladá, ako aj optimalizuje just-in-time kompilátor. Tento proces tiež umožňuje verifikáciu bezpečnosti CIL kódu počas behu, čím je dosiahnutá väčšia spoľahlivosť a bezpečnosť ako u strojového kódu.



Obr. 4.1: Prehľad .NET a Common Language Infrastructure

4.1.5 Knižnica tried .NET

Je ucelenou, objektovo-orientovanou zbierkou znovu-použiteľných typov, ktoré sú pevne integrované s CLR. Táto kolekcia pokrýva širokú škálu bežných programovacích potrieb, akými sú napríklad práca s reťazcami, zber dát, pripojenie k databáze, a prístup k súborom. Jej základ tvorí Base Class Library (BCL), čo sú menné priestory obsahujúce objekty a služby pre základnú funkcionalitu akou je napríklad práca so súborovým systémom, reťazcami, dátovými štruktúrami, vláknami a podobne. Nad týmto základom sa potom nachádzajú množiny tried pre špecifickejšie oblasti programovania (Databázy, XML, Web, GUI, Grafika).

4.2 Implementácia

Vytvorený program obsahuje viac ako 60 implementovaných tried. Nemalo by zmysel popisovať každú triedu samostatne, keďže je medzi mnohými uplatnený polymorfizmus a vzťahy dedičnosti. Ich účel a funkcionalita sú si podobné. Na nasledujúcich riadkoch bude snaha o postupný a prehľadný výklad podstatných častí implementácie.

4.2.1 Kreslené prvky pracovnej plochy

INamedObject

Rozhranie pre pomenované grafické objekty deklaruje nasledujúce vlastnosti: meno, viditeľnosť mena, obdĺžnik textu a správnosť definície konkrétneho objektu. Predpísaná je aj udalosť o zmene mena, keďže implementujúce grafické objekty nemajú možnosť kontrolovať svoju správnosť. Tieto objekty využívajú údaj o správnosti pre kreslenie červenej vlnovky pod textom.

DrawObject

Trieda DrawObject predstavuje hlavnú abstraktnú triedu pre všetky objekty diagramu prechodov. Jej hlavným rysom je metóda Draw, pomocou ktorej sa každý objekt vie vykresliť do predávaného grafického kontextu. Táto metóda je preťažená v dvoch formách ako verejná a chránená. Dôvodom je optimalizácia vykresľovania objektov. Verejná metóda prijíma booleovský argument o aktivite objektu. Ak tento hodnotou zodpovedá aktivite objektu, zavolá sa chránená metóda. Aktívne sú tie objekty, ktorých sa týka práve prevádzaná operácia ako napríklad vlastný pohyb či pohyb pripojeného objektu. Takto možno zefektívniť kreslenie vykreslením neaktívnych objektov do rastrového bufferu a počas pohybu prekresľovať len práve ovplyvňované objekty. Definované sú informácie [2] o tvare objektu. S týmto spoločným rysom súvisí verejná metóda InShape, zodpovedajúca dotaz na pozíciu objektu pod danými súradnicami. Vďaka tomu je možné nájsť objekt, ktorý sa práve nachádza pod kurzorom myši. Deklarovaná je metóda pre poskytnutie kontextového menu. Zaujímavá je tým, že v konkrétnych implementáciách objektov volá vždy rovnakú metódu objektu dodaného v argumente. Ako argument mu späť predáva inštanciu seba. Tým sa zabezpečí volanie správnej preťaženej metódy v objekte poskytujúcom kontextovo závislé menu, ktorá menu nastaví podľa stavu objektu. Táto myšlienka pochádza z návrhového vzoru Visitor a deleguje tak jedinú úlohu špecifickú pre každý zdedený objekt do separovanej triedy. Objekty triedy majú vlastnosť Owner, takisto typu DrawObject. Táto vyjadruje príslušnosť k inému objektu aby bolo známe napríklad ktorý objekt je zodpovedný za rušenie tejto inštancie. Príkladom je uzol krivky patriaci krivke. Zdedené triedy:

DrawState

Implementuje INamedObject. Disponuje zoznamom pripojených bodov hrán.

DrawNode

Predstavuje bod krivky hrany. Môže byť prepojený so stavom. Bod pozoruje pohybovú aktivitu svojho stavu odoberaním jeho udalostí. Ak je pripojený, riadi sa pohybom stavu. Ak je odpojený hýbe sa spolu s hranou.

DrawDefinition

Objekt definícia predstavuje textové pomenovanie hrany, čiže grafickú reprezentáciu definície pravidla. Implementuje INamedObject aby ju bolo možné premenovávať. Nesie v sebe objekt reprezentujúci túto definíciu podrobne (typ Rule).

DrawTransition

Predstavuje krivku tvoriacu hranu medzi stavmi. Objekt spravuje zoznam svojich bodov (typ DrawNode), ktoré definujú jej tvar a svoju definíciu (typ DrawDefinition). Implementuje algoritmus, ktorým vie určiť zaradenie nového bodu na správnu pozíciu v zozname svojich bodov. Algoritmus spočíva v postupnej interpolácii podskupín bodov a hľadaní odpovedajúceho segmentu krivky. Krivka pozná svoje stavy, ku ktorým je pripojená vďaka svojím krajným bodom. Od krajných bodov prijíma udalosť o zmene hostiteľského stavu. Vďaka tomu vždy okamžite vie či je na koncoch pripojená. Podľa tejto udalosti aktualizuje premennú s počtom chýb.

[2] objekty tried System.Drawing.Region a System.Drawing.Drawing2D.GraphicsPath

4.2.2 Logické prvky

Logické prvky abstrahujú definíciu a operácie automatu z konkrétnej implementácie simulátoru.

IState

Rozhranie IState deklaruje 4 vlastnosti stavu, ktorými sú booleovské hodnoty IsFinal, IsInitial, IsValid, a RealName.

ITape

Rozhranie pre vstupnú pásku automatu deklaruje jedinú metódu ShowNextToken, ktorá vracia symbol na pozícii čítacej hlavy.

IStack

Rozhranie pre zásobník deklaruje okrem hlavných metód Push, Pop a Peek naviac metódy IsEmpty a Configuration. Konfigurácia zásobníku je jedna zo súčastí kontroly prázdnych slučiek automatu počas simulácie.

Rule

Trieda pre pravidlo automatu kompletne definuje reláciu jedného prechodu. Pozná oba stavy (IState), ale aj odpovedajúci grafický objekt DrawTransition, s ktorým komunikuje. Definuje booleovskú vlastnosť Valid a počet vlastných chýb[3]. Stav chýb sleduje udalosťami od hrany a jej definície. Na zmenu definície reaguje tak, že jej nový reťazec okamžite podrobí syntaktickej analýze. Syntaktická analýza z reťazca vyberie 3 hodnoty: predchádzajúci vrchol zásobníku, zmenu na zásobníku a vstupný symbol. Podľa výsledku nastaví platnosť definície, aktualizuje počet chýb a uloží si výsledky analýzy. Zainteresované objekty tiež upozorňuje na zmeny udalosťou.

PdaLogic

Táto trieda predstavuje mechanizmus činnosti automatu. Využíva práve vyššie uvedené logické rozhrania a množinu pravidiel. Pracuje takmer samostatne, pri čom verejne zverejňuje len metódy pre inicializáciu, krok simulácie, zastavenie, voľbu pravidla a test determinizmu. Výpočtový krok automatu prevádza podľa možnosti naraz, pričom o jednotlivých operáciách informuje simulátor vyvolaním svojej udalosti. Keďže pohyb hlavy nie je v rozhraní pásky deklarovaný, samotná logika neposúva hlavu, ale len vysiela požiadavku časovanej simulácii. Test determinizmu sa vykonáva prechodom cez kolekciu s pravidlami, pri čom sa vytvára komplexná štruktúra generických hašovacích tabuliek a zoznamov o troch úrovniach, ktorá umožňuje prehľadné a efektívne vyhladávanie.

4.2.3 Editačné príkazy

Command

Je abstraktnou triedou pre objekty operácií editoru. Objekty zdedené z Command zapuzdrujú špecifické informácie potrebné k zvráteniu a opätovnému vykonaniu úkonu. Trieda pre ne definuje základ metód Undo a Redo. Statická časť tejto triedy má na starosti správu vzniknutých objektov v dvoch

[3]Tie môžu byť až 4: chýbajúci počiatočný stav, chýbajúci cieľový stav, nesprávna definícia vstupného symbolu a nesprávna definícia predchádzajúceho vrcholu na zásobníku.

zásobníkoch. Po prvotnom vykonaní sa špecifické príkazy Command ukladajú na Undo zásobník. Pri spätnej operácii sa vyberie objekt z vrcholu zásobníku a zavolá sa jeho vlastná Undo metóda. Následne sa vloží na zásobník Redo. Pri vkladaní novej operácie na Undo zásobník je Redo zásobník vyprázdnený. Vykonávanie operácií je oznamované udalosťou, aby bolo možné nastavovať Dirty bit, ktorým sú kontrolované zmeny od posledného uloženia. Konkrétne triedy dediace z Command:

- AddTransitionCommand
- AddNodeCommand
- AddStateCommand
- FinalStateCommand
- InitialStateCommand
- DeleteObjectCommand
- DeleteSelectionCommand
- MoveCommand
- MoveSelectionCommand
- RemoveNodeCommand
- RenameCommand
- ReverseTransitionCommand

4.2.4 Komponenty

EditorArea

Je to komponent zdedený z triedy System.Windows.Forms.Panel. Umožňuje tak prijímať užívateľský vstup napríklad od kurzoru myši. Disponuje sadou editačných nástrojov, z ktorých je vždy jeden aktívny. Aktívnemu nástroju sa podľa potreby predáva zodpovednosť za obsluhu vstupu od užívateľa, alebo grafický kontext pri kreslení. Každý nástroj má tak možnosť v čase svojej aktivity kresliť do tejto plochy a prijímať všetky udalosti myši. Komponent obsahuje jedinú inštanciu triedy PdaSimulator, nad ktorou operuje spolu s nástrojmi na editáciu. Samotná plocha na sebe vykresľuje obsah simulátoru, k čomu používa aj rastrový buffer (bitovú mapu) pre optimalizovanie kreslenia. Do tohto rastru vykresľuje objekty, ktorých vzhľad nie je počas operácií ovplyvnený. Tým odpadá nutnosť pri každom prekreslení vykresľovať opakovane všetky objekty simulátoru a raster sa použije ako podklad na začiatku kreslenia. Plocha v dobe nečinnosti programu kontroluje priestor zaberaný grafickými objektmi simulátoru a podľa potreby mení svoju veľkosť.

InputTape

Vstupná páska je komponent zložený z dvoch tlačidiel, panelu a textového pola. Textové pole jej slúži pre vstup a panel pre grafický výstup animácie. Tlačidlá umožňujú plynulé prevíjanie pásky. Implementuje rozhranie ITape. Okrem toho vystavuje verejne len metódy Load (príkaz pre prepnutie do režimu simulácie), Disable a MoveHead. Animácia je riadená dvoma časovačmi. Jeden slúži na pohyb hlavy a druhý na centrovanie pásky. Pri požiadavke na posuv hlavy sa najprv páska podľa

potreby sama pretočí tak, aby bola viditeľná hlava a následne sa spustí pohyb hlavy. Komponent pásky umožnuje aj pohyb hlavy vlavo, ale táto možnosť sa nevyužíva. Jednoduchým rozšírením by bolo umožnenie zápisu na pásku.

StackListBox

Trieda dedí z komponentu užívateľského rozhrania System.Windows.Forms.ListBox a pridáva rozhranie IStack, ktorým sa ovláda.

RulesListBox

Jej kolekcia položiek je množinou pravidiel automatu. Táto trieda implementuje vlastné vykresľovanie pravidiel. Tie jej sprostredkovávajú informácie o zmenách na editačnej ploche a opačným smerom umožňujú selekciu hrán prostredníctvom zoznamu.

4.2.5 Simulátor

PdaSimulator

Táto trieda má v programe dve hlavné funkcie: spravovanie grafických objektov a samotnú simuláciu. Je zdedená z parametrizovanej generickej triedy Collection<DrawObject> a funguje tak ako kolekcia všetkých kreslených objektov.

Má definované vlastné metódy pre pridávanie resp. odoberanie prvkov. Stavy napríklad pridáva pod prechody tak, aby nemohli zakrývať iné objekty. V týchto metódach sa ale predovšetkým prihlasuje resp. odhlasuje z odoberania udalostí týchto objektov. Môže tak dozerať na celkový počet chýb v navrhnutom modeli a teda nepovoliť spustenie simulácie. Umožňuje vykreslenie celého prechodového diagramu iteráciou nad vlastnou inštanciou. Ďalej napríklad spravuje zoznamy označených a zvýraznených objektov. Na rozdiel od grafickej definície prechodu (DrawDefinition), ktorú pri zmene kontrolovala trieda pravidla (Rule), na zmeny v názvoch stavov reaguje práve simulátor. Hašovou tabuľkou účinne kontroluje duplicitu v pomenovaniach stavov a podľa nej nastavuje ich správnosť či nesprávnosť. Simulátor funguje ako fyzický prostredník medzi oddelenými prvkami, keďže zabezpečuje ich súhru. Disponuje referenciami na komponenty zásobníku, pravidiel, vstupnej pásky a na tlačidlá ovládania v nástrojovej lište. Obsahuje objekt logiky automatu, ktorá ich vidí ako spomenuté logické typy. Je ale nutné zdôrazniť, že samotná logika nemení ich stav keďže prevádza celý výpočet naraz a simulácia vyžaduje časovanie. Namiesto toho vysiela o svojich operáciách správy (typ SimulationLogicEventArgs) prostredníctvom udalostí. Správy sú rozlíšené vymenovaným typom a obsahujú objekty súvisiace s prevedenou operáciou. Simulátor tieto správy prijíma a ukladá do frontu. Ak simulácia ešte nebeží, príjem správy spustí časovač, ktorý v nastavenom intervale vyberá správy a interpretuje si ich ako príkazy. Pri vykonávaní príkazu správy sa podľa jej obsahu zostaví aj správa pre užívateľa. Pre nemenné správy sa ako slovník použije statické pole reťazcov triedy SimulationLogicEventArgs indexované hodnotou vymenovaného typu. Správy simulátoru potom prijíma formulár okna, ktorý ich užívateľovi zobrazuje na výstupe. Pri správe o možnosti výberu z nedeterministických pravidiel je naviac vyslaná požiadavka na sledovanie užívateľovho vstupu. Túto zachytí a obslúži editačná plocha. Vyprázdnením frontu sa časovač zastaví. Zastavením časovača je realizovaná aj pauza simulácie. Príkazy vo fronte sa tak nevykonajú až do opätovného spustenia, prípadne sa front vyprázdni zastavením. Na záver popisu tejto triedy ešte dodajme, že sa tu nachádzajú metódy Save a Load, ktorými sa za pomoci triedy PdaSimulatorStructure zostaví a uloží resp. načíta a rekonštruuje objekt opisujúci automat.

4.2.6 Nástroje

Nástroje slúžia na vykonávanie špecifických akcií užívateľa na editačnej ploche s ktorou spolupracujú. Niektoré nástroje majú viac funkcií, ktoré závisia od ich použitia alebo od stavu konkrétneho objektu. Vykonané operácie dokážu zaznamenať a vytvoriť s nimi odpovedajúci objekt typu Command.

Tool

Táto abstraktná trieda deklaruje metódy pre obsluhu udalostí myši a kláves. Tieto zodpovedajú metódam editačnej plochy. Plocha tak deleguje časť obsluhy týmto nástrojom. Rovnako je umožnené kreslenie do grafického kontextu, ktoré tu ale využíva len nástroj pridávajúci pravidlo medzi stavy. Zdedenými triedami sú:

- AddStateTool,
  - InitStateTool,
  - FinalStateTool,
- AddTransitionTool,
- DeleteTool,
- PointerTool,
- RenameTool.

4.2.7 Nastavenia programu

EditorSettings

Trieda užívateľských nastavení aplikácie je vytvorená podľa vzoru Singleton, čo znamená sa v programe chová ako statický globálny objekt. Princíp návrhového vzoru spočíva v definovaní privátneho konštruktoru, čím sa zamedzí použitiu implicitného verejného konštruktoru. Trieda obsahuje statickú privátnu premennú s vlastnou inštanciou. Takúto inštanciu tak môže vytvoriť len vlastná metóda. Táto jediná inštancia je sprístupnená statickou metódou Instance. Trieda EditorSettings implementuje tzv. lenivú inicializáciu, kedy sa táto inštancia vytvorí až pri prvom volaní metódy Instance. Vystavená je statická udalosť SettingsChanged. K nej sa registrujú zainteresované triedy vo svojich konštruktoroch objektov alebo statických konštruktoroch tried. Statický konštruktor je využitý napríklad v triede DrawObject kde sa nachádzajú spoločné perá a štetce pre kreslenie objektov. Výhodou vzoru Singleton je práve skutočná inštancia triedy ukrytá za metódou a teda možnosť jej zmeny počas behu programu. Z tohto dôvodu bola pre túto triedu implementovaná súkromná metóda Clone, ktorá duplikuje inštanciu objektu. Klonovaná inštancia sa volá substitutor a rovnako staticky ju zverejňuje podľa potreby metóda Substitutor . Účelom tohto dočasného náhradníka je umožnenie práce s dialógom nastavení (OptionsDialog) bez nutností zmien. Znamená to, užívateľ môže napríklad náhradníka nastaviť, alebo aj uložiť, či načítať zo súboru bez nutnosti zmeny používanej inštancie.

Kapitola 5

Práca so simulátorom

Táto kapitola popisuje užívateľskú stránku vytvorenej aplikácie a možnosti, ktoré implementuje.

5.1 Editácia automatu

Model automatu je možné začať vytvárať okamžite po spustení programu. Editáciu umožňuje prvých šesť prvkov z nástrojovej lišty, viz obrázok 5.1. Ako prvý musí byť definovaný aspoň jeden stav, aby bolo možné definovanie pravidla prechodu. Táto hrana sa dá vytvoriť jedine medzi existujúcimi stavmi. Viacnásobné označenie objektov je možné za pomoci klávesy Ctrl. Vyznačené objekty je možné mazať klávesou Delete.

Obr. 5.1: Prvky nástrojovej lišty

Stavy

Pre umiestnenie nového stavu možno použiť tri nástroje:

- *Nový stav* slúži len na vytvorenie stavu.
- *Počiatočný stav* na prázdnej ploche vytvorí nový počiatočný stav. Pri použití na existujúci stav sa tento nastaví ako počiatočný resp. zruší tento atribút, ak už bol nastavený. Editor povoluje len jediný počiatočný stav podľa definície zásobníkového automatu
- *Koncový stav* rovnako ako počiatočný vytvára nový stav alebo prepína vlastnosť existujúceho stavu.

Pravidlá

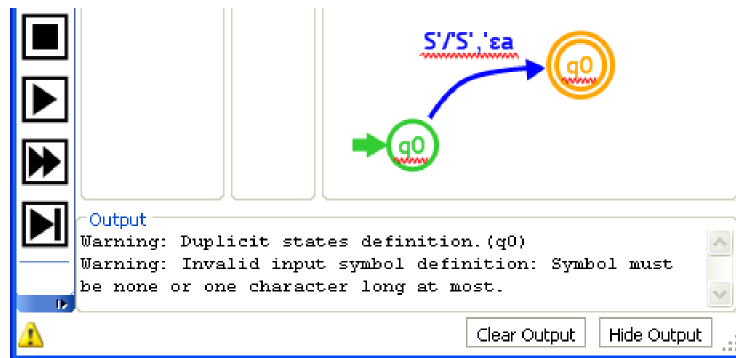
Krivka pravidla sa vytvára aktivovaním vlastného nástroja na lište a spôsobom ťahaj a pusť medzi zvolenými stavmi. V prípade rovnakého stavu sa z krivky vytvorí základná slučka zo štyroch bodov. Nové body sa dajú pridávať resp. rušiť dvojitým kliknutím do tvaru krivky resp. existujúceho bodu (pri použití základného nástroja kurzoru). Vytvorenú krivku je možné od stavu odpojiť ťahom jej koncového bodu a podľa potreby presmerovať do iného stavu. Pustenie odpojenej krivky na plochu je tiež možné a to z praktických dôvodov, aby program užívateľa zbytočne neobmedzoval.

Premenovanie

Popisy jednotlivých objektov je možné meniť priamo na ploche a to dvojitým kliknutím na danú definíciu pri použití základného nástroja kurzoru.

Premenovanie definície pravidla

Grafická reprezentácia pravidla automatu je prevzatá z predmetu Formálne jazyky a prekladače a sa skladá z troch logických častí. Pre rozoznanie používaných oddeľovacích znakov lomítko a čiarka sa používajú apostrofy, takže predpis definície pravidla má tvar “(1)’/’(2)’,’(3)”, kde (1) udáva pôvodný vrchol zásobníku, (2) udáva zmenu na vrchole zásobníku a (3) je vstup čítaný zo vstupnej pásky. Symbol “ε” je možné zadať v tvare “(e)”, alebo prázdnym (2) resp. (3). Pri chybnom zadaní sa vypíše varovanie a definícia sa podčiarkne červenou vlnovkou 5.2.



Obr. 5.2: Príklad varovania pred chybou

Odstránenie

Samostatný nástroj na mazanie umožňuje rýchle odstraňovanie objektov. Grafické prvky pod kurzorom sú pri pohybe zvýraznené. Jeho použitie je vhodné najmä pri väčšom počte nežiaducich objektov.

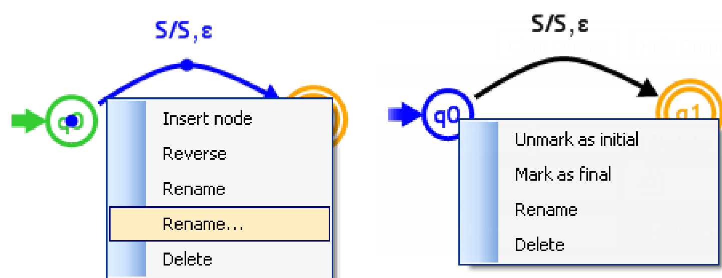
Obr. 3 Príklad varovania pred chybou

Kontextové menu

Plocha editoru obsahuje menu, ktoré ponúka v závislosti od kontextu kurzoru rovnaké možnosti ako nástrojová lišta až na vytvorenie nového prechodu medzi stavmi. Príklad je uvedený na 5.3. Pre objekt objekt prechodu a jeho definície ponúka menu naviac možnosť vyvolať *dialóg premenovania definície*.

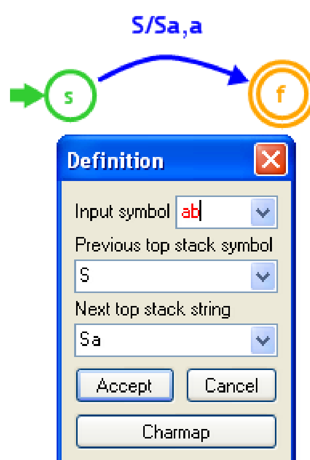
Dialóg definície pravidla

Pre jednoduchšie vytvorenie definície pravidla slúži modálny dialóg definície pravidla zobrazený na obrázku 5.4. K dispozícii sú tu tri combo-box prvky pre samostatné zadávanie (1), (2) a (3). Pri chybnom vstupe sa farba textu zmení na červenú, aby tak bola užívateľovi chyba hneď oznámená. Ponuka combo-boxov obsahuje tie príslušné časti definícií, ktoré boli vytvorené užívateľom počas predchádzajúcej editácie. Výber z týchto použitých reťazcov sa automaticky ponúka zo zoznamu pri zadávaní vstupu. Dialóg navyše obsahuje tlačidlo pre vyvolanie aplikácie Charmap. Dôvodom pre



Obr. 5.3: Príklad kontextového menu stavu a hrany

existenciu tohto dialógu je uľahčenie zmeny definície so súčasnou validáciou a umožnenie zadania špeciálnych znakov nenachádzajúcich sa na klávesnici.



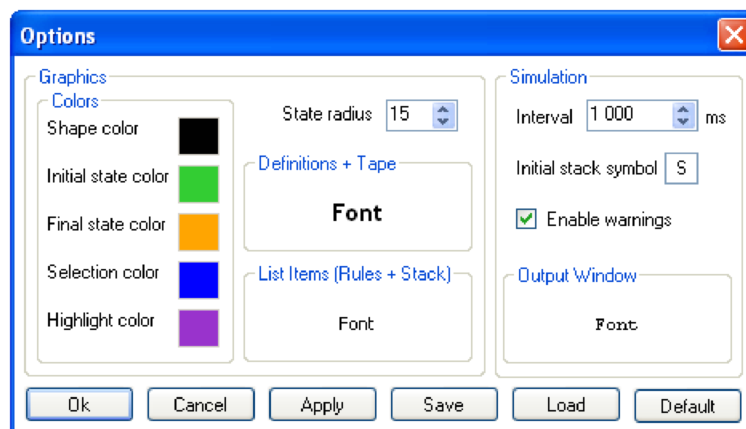
Obr. 5.4: Dialóg pre definíciu pravidla

Nastavenia

Nastavenia sú vytvorené s ohľadom na demonštračné účely tohto nástroja. Umožňujú prevažne zmenu vzhľadu editoru, tak aby bolo možné zlepšiť viditeľnosť jednotlivých prvkov. Ďalej v nich možno nájsť nastavenie intervalu pre operáciu automatu, počiatočný symbol pre zásobník a možnosť vypnutia varovaní od simulátoru. Nastavenia je tiež možné ukladať do súboru. Dialóg je zobrazený na obrázku 5.5

5.2 Simulácia

Simulácia je možná len na formálne správnom modeli zásobníkového automatu a so zadanou metódou prijatia, tj. koncovým stavom, prázdnym zásobníkom alebo oboma súčasne.Na prípadnú chybnosť je užívateľ upozornený simulátorom. K ovládaniu slúžia ovládacie prvky v nástrojovej lište. Je možné previesť jeden animovaný krok automatu, ale aj spustiť automatickú simuláciu, ktorú je možné kedykoľvek pozastaviť. Ovládanie ponúka aj možnosť skoku, ktorý sa zastaví na konci simulácie alebo pri možnosti výberu z viacerých prechodov. Počas celej simulácie vypisuje simulátor



Obr. 5.5: Nastavenia programu

do okna výstupu hlásenia o prevedených operáciách automatu, viz 5.6. Pri konfrontácii s nedeterminizmom sa simulácia zastaví a vyzve užívateľa k voľbe z ponúknutých možných prechodov. Voľba sa vykoná kliknutím na príslušnú zvýraznenú hranu. Z chyby alebo neúspešnej simulácie sa simulátor dokáže zotaviť a pokračovať po náprave ale len v prípade, že sa nevyprázdnil zásobník.

Vstupná páska

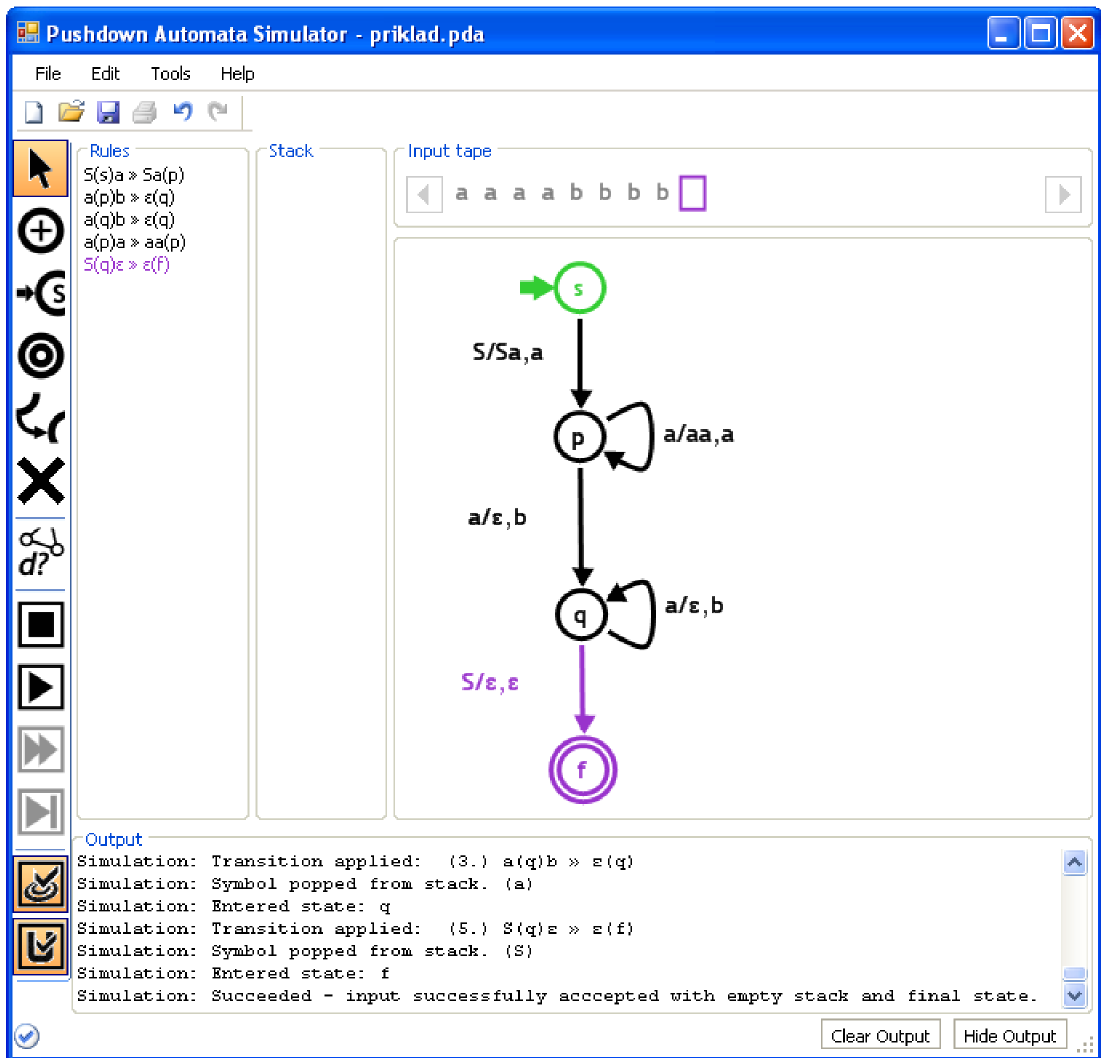
Počas editácie funguje komponent pásky ako vstupné pole pre text. Pri začiatku simulácie sa vstupné pole zablokuje a zadaný vstup sa vykreslí ako páska so symbolmi. V prípade vstupu dlhšieho ako je viditeľná časť, sú k dispozícii tlačidlá pre jej prevíjanie. Ak je pri simulácii páska pretočená tak, že sa hlava nachádza mimo viditeľnú časť, simulátor najprv vycentruje časť pásky na ktorej sa nachádza hlava a až potom prečíta ďalší vstupný symbol. Tento interval nesúvisí s údajom v nastaveniach a nedá sa meniť.

Test determinizmu

Testom je možné kontrolovať len formálne správne vytvorený automat. Výsledok testu sa zobrazuje v okne výstupu. Ak je testovaný automat nedeterministický, na výstup sú vypísané pravidlá, ktoré porušujú determinizmus a zároveň sa vyznačia na ploche.

Poznámka

Pri vytváraní automatu je užívateľovi umožnené oddeľovať hrany od uzlov grafu ako aj rušenie pripojených uzlov - stavov automatu a to aj v priebehu simulácie. Zodpovednosť za správnosť činnosti automatu tak zostáva výlučne na užívateľovi. Toto je umožnené z praktických dôvodov. Pri takejto operácii sa samozrejme stáva automat formálne neplatným a simulácia sa v takom prípade nespustí. Pri spustenej simulácii sa zmeny vyhodnotia až v nasledujúcom kroku automatu, kedy sa simulácia ukončí alebo pokračuje so zmenami.



Obr. 5.6: Be

Kapitola 6

Záver

Táto práca sa zaoberala návrhom a implementáciou systému, ktorý by umožňoval simuláciu činnosti zásobníkového automatu. Hlavnou požiadavkou na aplikáciu bolo kvalitné užívateľské rozhranie, keďže podobné dostupné nástroje bývajú málo názorné, či ťažko ovládateľné. Zároveň bola snaha o vytvorenie rovnakej grafickej reprezentácie, aká je použitá v prednáškach predmetu Formálne jazyky a prekladače. Tieto požiadavky na systém sa podarilo splniť hlavne vďaka podrobnej analýze a návrhu, ktoré sa ukázali ako nutné súčasti vývoja. Tieto etapy, ktoré predchádzali samotnú realizáciu tak následne umožnili jej hladký priebeh bez výrazných problémov. V tejto navrhnutej koncepcii takto našlo uplatnenie hneď niekoľko návrhových vzorov. Ich užitím sa zároveň výrazne umocnili možnosti ďalšieho rozšírenia. Možné by bolo pridanie ďalších modelov alebo ďalších pások či zásobníkov. Keďže program ale nebol navrhovaný priamo s úmyslom jeho budúceho vývoja, tak by si takáto iniciatíva vynútila niektoré zmeny. Zmena by sa napríklad odrazila na generalizácii niektorých rozhraní či tried (napr. EPdaSimulator, TMSimulator,.. ako Simulator) a ďalšom uvoľnení väzieb medzi triedami. Prípadne by tak možno našiel využitie aj niektorý továrenský návrhový vzor. Možnosťou by bolo aj pripojenie nejakého lexikálneho analyzátoru pomocou adaptérového vzoru, ktorý by už konkrétnejšie rozpoznával slová určitého jazyka. Nenáročné by bolo napríklad vytvorenie možnosti pre výber, prípadne špecifikovanie vlastného formátu notácie (napr. podľa predmetu Teoretická informatika), či ukladanie v ďalších formátoch. Týmto programom by som chcel v jeho užívateľoch podnietiť záujem o problematiku formálnych jazykov umožnením vytvárania a simulovania vlastných modelov. Práca na tomto projekte mi dala hlavne nové znalosti o objektovo-orientovanom návrhu a motiváciu k ďalšiemu štúdiu tejto oblasti informatiky.

Literatúra

[1] V. Claus and A. Schwill. *Lexikón Informatiky*. Slovenské pedagogické nakladateľstvo, Bratislava, Slovak Republic, 1991. ISBN 80-08-00755-9.

[2] Microsoft Corporation. Frequently Asked Questions About Visual C#, 2007. [Online; dostupné 20. 1. 2007].

[3] Microsoft Corporation. .NET Framework Conceptual Overview, 2007. [Online; dostupné 20. 1. 2007].

[4] H. Lam and T. L. Thai. *.NET Framework Essentials*. O'Reilly, 2001. 0-596-00165-7.

[5] A. Meduna. *Automata and Languages*. Springer-Verlag, London, Great Britain, 2000. ISBN 1-85233-074-0.

[6] D. A. Simovici and R. L. Tenney. *Theory of Formal Languages With Applications*. World Scientific Publishing Company, 1999. ISBN 981-02-3729-4.

[7] Wikipedia. Noam Chomsky — Wikipedia, The Free Encyclopedia, 2007. [Online; dostupné 20. 1. 2007].

[8] M. Češka and Z. Rábová. *Gramatiky a jazyky*. Vysoké Učení Technické v Brne, Brno, Czech Republic, fourth edition, 1992. ISBN 80-214-0449-3.