SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Softvérové inžinierstvo, Informačné systémy

---

AUTOMATY

Projektová dokumentácia

Tím číslo 2

Vedenie projektu: Mgr. Daniela Chudá, PhD.

19.5.2009

Bc. Robin Bábíček
Bc. Matúš Coranič
Bc. Matúš Čelko
Bc. Celestín Černák
Bc. Daniela Miloňová
Bc. Katarína Poláková

Projektová dokumentácia – časť I

INŽINIERSKE DIELO

OBSAH

ZADANIE PROJEKTU

Teóriu formálnych jazykov a automatov uviedol Noam Chomsky ako časť teórie počítačovej vedy už v roku 1955. Moderné aplikácie formálnych jazykov a automatov sa objavujú najmä v oblasti tvorby návrhu kompilátorov a popise abstraktných výpočtových zariadení. Študenti informatiky na celom svete sa stretávajú v bakalárskom štúdiu s výukou formálnych jazykov a automatov, ktorá je nezriedka vedená tradične bez podpory simulátorov. Pri štúdiu formálnych výpočtových modelov sa vyskytne množstvo otázok: "Ako vysvetliť či znázorniť funkcionalitu a činnosť automatu - abstraktného výpočtového zariadenia?", "Ako prepojiť informácie medzi matematickým zápisom zariadenia a funkcionalitou výpočtového zariadenia predstavujúcou sekvenčný proces, ako prepojiť stavový diagram s prechodovou funkciou a vstupom?" "Ako vizualizovať nedeterminizmus výpočtových zariadení?".

Vytvorte integrovaný nástroj pre simuláciu, vizualizáciu a testovanie rôznych výpočtových zariadení, ako konečný automat, zásobníkový automat, Turingov stroj, RAM, počítadlový stroj, generátor gramatík. Nástroj by mal umožňovať:

- prácu s preddefinovanými výpočtovými zariadeniami,
- rozšírenie o prácu s operáciami nad výpočtovými zariadeniami,
- definovanie svojich vlastných zariadení,
- prácu s nedeterministickými výpočtovými zariadeniami,
- možnosť testovania študentov, generovanie problémových situácii nad zariadením aj s kontrolným riešením,
- jednoduché rozhranie a ovládanie, export a import jednotlivých výpočtových zariadení, zachovanie bezpečného prístupu k informáciám o testovaní vedomostí študentov, modularita a rozšíriteľnosť.

Naši študenti využívajú pri štúdiu rôzne parciálne simulátory, naprogramované na iných univerzitách či naprogramované priamo našimi študentmi. Práca so simulátormi patrí k najobľúbenejším činnostiam v predmete. Nástroj pokrývajúci široké spektrum problematiky formálnych jazykov a automatov je JFLAP .

SLOVNÍK POJMOV

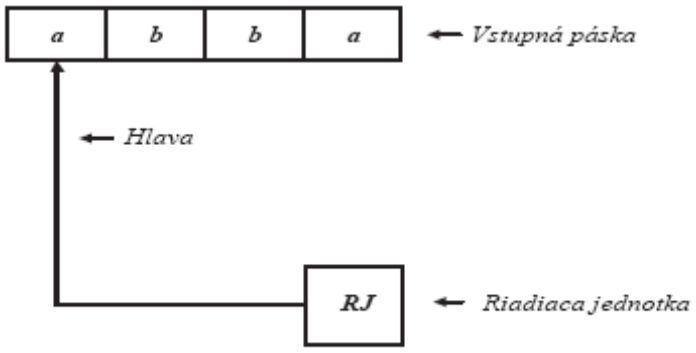
- *determinizmus* - je formálna abstrakcia takých výpočtov, v ktorých je jednoznačne určený nasledujúci krok
- *nedeterminizmus* – je formálna abstrakcia takých výpočtov, v ktorých nie je jednoznačne určený nasledujúci krok
- *páska* – vstupné médium automatu. V závislosti od typu automatu je v jednom alebo v obidvoch smeroch nekonečná. V každom poli pásky je zapísaný symbol
- *hlava* – je umiestnená nad konkrétnym poľom pásky, v každom kroku hlava číta alebo zapisuje symbol a posunie se doľava alebo doprava, prípadne zostane na mieste kde sa práve nachádza
- *stavový diagram* – grafické zobrazenie automatu, zobrazujúce prechod medzi jednotlivými stavmi. Stavy sú graficky reprezentované kružnicami a sú jednoznačne označené. Konečné stavy sú zobrazené dvoma sústrednými kružnicami.
- *prechodová funkcia* – analytické vyjadrenie prechodovej charakteristiky
- *zásobník* – je všeobecná dátová štruktúra (tzv. abstraktný dátový typ) používaná pre dočasné ukladanie dát. Pre zásobník je charakteristický spôsob manipulácie s dátami - dáta uložené ako posledné budú čítané ako prvé. Preto sa používá tiež výraz LIFO z anglického „*Last In – First Out*“.
- *pamäť* – pole registrov
- *register* – najmenšia adresovateľná pamäťová jednotka, ktorej obsahom je celé číslo

1. ANALÝZA PROBLÉMU

V tejto kapitole sa zaoberáme analýzou problematiky automatov z pohľadu ich definície a základných vlastností. Táto časť predstavuje základný teoretický úvod do problematiky a jej preštudovanie je odporúčané na pochopenie nasledujúcich častí. Na základe tejto analýzy a prehľadu existujúcich riešení sme vypracovali špecifikáciu požiadaviek.

1.1. KONEČNÝ AUTOMAT

Konečný automat (angl. *finite automaton*, FA) je teoretický výpočtový model používaný v informatike na štúdium formálnych jazykov. Predstavuje jednoduchý počítač, ktorý sa môže nachádzať v jednom z niekoľkých definovaných stavov, pričom medzi jednotlivými stavmi prechádza na základe symbolov, ktoré číta na vstupe. Automat pozostáva z riadiacej jednotky, vstupnej pásky a čítacej hlavy (Obr. 1)



Obr. 1: Schéma konečného automatu

Rozlišujeme 2 základné druhy konečných automatov:

- Deterministické
- Nedeterministické

Keďže triedy jazykov, ktoré oba typy rozpoznávajú sú ekvivalentné, je možné ku každému nedeterministickému konečnému automatu zostrojiť ekvivalentný deterministický konečný automat.

1.1.1. DETERMINISTICKÝ KONEČNÝ AUTOMAT

Deterministický konečný automat (DFA) je formálne definovaný ako pätica
A = (K, Σ, δ, q_0, F), kde:

K je konečná množina stavov

Σ je vstupná abeceda

δ je prechodová funkcia, pričom platí $\delta: K \times \Sigma \rightarrow K$

q_0 je počiatočný stav $q_0 \in K$

F je množina koncových stavov $F \subseteq K$

1.1.2. NEDETERMINISTICKÝ KONEČNÝ AUTOMAT

Nedeterministický automat (NFA) sa líši iba v špecifikácii prechodovej funkcie:

δ je prechodová funkcia, pričom platí: $\delta : K \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^K$

Tento automat nemusí mať v každom kroku jednoznačne definovaný stav, ktorým má pokračovať. Znamená to, že z určitého stavu je možné prejsť do viacerých stavov za rovnakej podmienky. Riešení teda môže byť viacero. Voľba cesty je ponechaná na používateľovi automatu, alebo automat vygeneruje všetky možnosti, z ktorých sa vyberie jedna najlepšia.

Reprezentácia

Konečný automat môžeme reprezentovať:

- stavovým diagramom,
- formálnym zápisom,
- maticou prechodových funkcií (pre účely simulátora neuvažujeme).

Jednotlivé možnosti reprezentácie automatu si ukážeme na jednoduchom príklade. Popis automatu formálnym zápisom vyzerá nasledovne:

Nech **A1 = (K, Σ, δ, q_0, F)**, *kde:*

$K = \{q_0, q_1\}$,

$\Sigma = \{a,b\}$,

$\delta(q_0,a) = q_0$

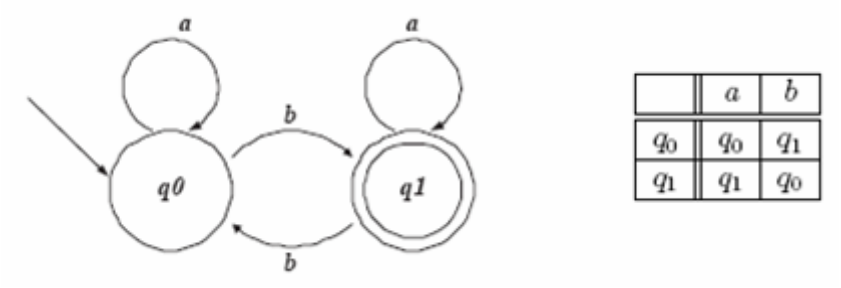
$\delta(q_0,b) = q_1$

$\delta(q_1,a) = q_1$

$\delta(q_1,b) = q_0$

$F = \{q_1\}$

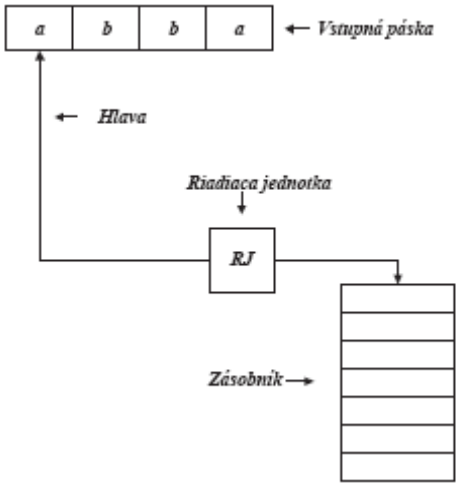
Reprezentácia tohto automatu stavovým diagramom a maticou prechodových funkcií je zobrazená na Obr. 2.



Obr. 2: Reprezentácia konečného automatu

1.2. ZÁSOBNÍKOVÝ AUTOMAT

Zásobníkový automat (angl. *push-down automaton*, PDA) je teoretický výpočtový model, ktorý rozpoznáva slová bezkontextového jazyka. PDA je v zásade konečný automat, ktorý má pridanú abstraktnú údajovú štruktúru zásobník. Jeho schéma je zobrazená na Obr. 3



Obr. 3 Schéma zásobníkového automatu

Pri manipulovaní so zásobníkom budeme používať nasledujúce operácie:

- **push** – pridanie prvku na vrch zásobníka
- **pop** – vybratie prvku z vrchu zásobníka

- **skip** - preskočenie, žiadna operácia
- **top** – prečítanie prvku na vrchu zásobníka
- **empty** – test na prázdnosť zásobníka

Rovnako ako v prípade FA rozlišujeme 2 základné druhy zásobníkových automatov:

- Deterministické (DPDA)
- Nedeterministické (NPDA)

V tomto prípade ale triedy jazykov rozpoznávaných DPDA a NPDA nie sú ekvivalentné. Neexistuje teda spôsob ako ku každému NPDA zostrojiť príslušný DPDA.

1.2.1. DETERMINISTICKÝ ZÁSOBNÍKOVÝ AUTOMAT

Deterministický zásobníkový automat je formálne špecifikovaný ako sedmica:
A = (K, Σ, Γ, δ, q_0, Z_0, F), kde:

K je konečná množina stavov

Σ je vstupná abeceda

Γ je zásobníková abeceda

δ je prechodová funkcia, pričom platí $\delta: K \times \Sigma \times \Gamma \rightarrow K \times \Gamma^{\bullet}$

q_0 je počiatočný stav $q_0 \in K$

Z_0 je počiatočný zásobníkový symbol $Z_0 \in \Gamma$

F je množina koncových stavov $F \subseteq K$

Podobne ako u konečných automatov aj zásobníkový automat môže mať len jeden počiatočný stav. Maximálny počet koncových stavov nie je určený.

Zásobníkový automat je deterministický, ak platia nasledovné podmienky:
$\forall q \in K, \forall a \in \Sigma, \forall Z \in \Gamma$:

1. množina δ(q, a, Z) obsahuje najviac jeden prvok
2. množina δ(q, ε, Z) obsahuje najviac jeden prvok
3. ak množina δ(q, ε, z) nie je prázdna, potom množina δ(q, a, Z) je prázdna

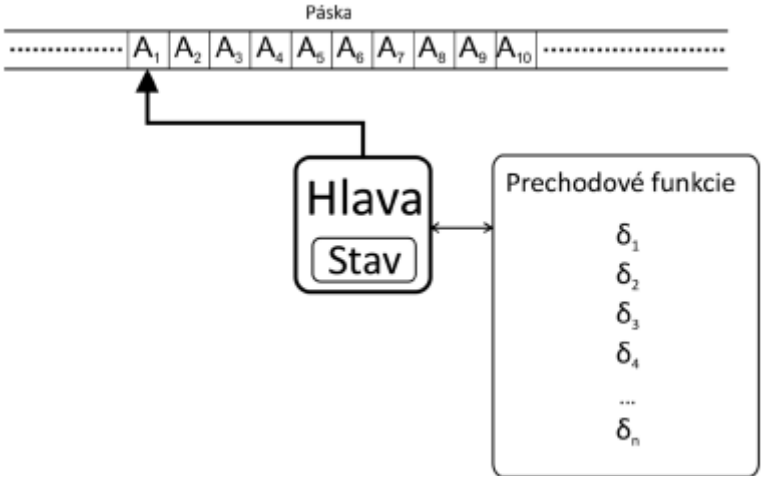
1.2.2. NEDETERMINISTICKÝ ZÁSOBNÍKOVÝ AUTOMAT

Definícia nedeterministického zásobníkového automatu sa opäť líši v špecifikácii prechodovej funkcie:

δ je prechodová funkcia, pričom platí: $\delta : K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{K \times \Gamma^*}_{KON}$

1.3. TURINGOV STROJ

Turingov stroj (angl. *Turing machine*, TM) je teoretický výpočtový model definovaný matematikom Alanom Turingom. Definovať ho môžeme ako konečný automat spojený s externým úložiskom alebo pamäťovým médiom. Schéma je znázornená na Obr. 4.



Obr. 4: Schéma Turingovho stroja

Turingov stroj je definovaný ako usporiadaná šestica **T (K, Σ, Γ, δ, q_0, F)**, kde:

K je konečná množina prvkov

Σ je vstupná abeceda

Γ je zásobníková abeceda

δ je prechodová funkcia, pričom platí

q_0 je počiatočný stav

F je množina koncových stavov

Konfigurácia

Konfiguráciu Turingovho stroja T popisujeme ako trojicu **T (q, ~ , i)**. Kde:

- $q \in K$ je aktuálny stav T
- ~ je postupnosť symbolov $(P - \{B\})^*$ a predstavuje neprázdnu časť z pásky

- i je celé číslo vyjadrujúce vzdialenosť hlavy stroja T od začiatku ~

Prechodová funkcia

Prechodovú funkciu definujeme takto. Nech $(q, A_1A_{2,} \ldots A_{\sim} \ i)$ je konfigurácia T, kde $1 \le i \le n+1$.

Ak $1 \le i \le n+1$ a $\delta(q, A_I) = (p, A, R)$,

potom $(q, A_1 A_{2,} \ldots A_{\sim} \ i) \sim (q, A_1 A_{2,} \ldots A_{I-1} A A_{I+1} \ldots A_{\sim} \ i + 1)$.

T teda vypíše symbol a pohne sa doprava.

Ak $2 \le i \le n+1$ a $\delta(q, A_I) = (p, A, L)$,

potom $(q, A_1 A_{2,} \ldots A_{\sim} \ i) \sim (q, A_1 A_{2,} \ldots A_{I-1} A A_{I+1} \ldots A_{\sim} \ i - 1)$.

T teda vypíše symbol a pohne sa doľava.

Výpočet

Výpočet Turingovho stroja začína v určenom počiatočnom stave na určenom mieste na páske, na ktorej sa nachádza vstupné slovo. Následne sa opakuje postupnosť týchto krokov:

1. Ak je aktuálny stav z množiny koncových stavov, výpočet je dokončený (môže sa povedať, že Turingov stroj akceptoval vstupné slovo).
2. Čítacia hlava prečíta symbol na páske, nad ktorým sa nachádza.
3. Ak existuje prechodová funkcia vyhovujúca vstupným požiadavkám – prečítaný symbol a aktuálny stav, vykoná sa:
    a. Zmení sa stav podľa príslušnej prechodovej funkcie.
    b. Zapíše sa znak na aktuálnu pozíciu.
    c. Hlava sa pohne vľavo alebo vpravo (niektoré modifikácie modelu umožňujú definovať aj prechodové funkcie bez pohybu hlavy).
4. Ak neexistuje prechodová funkcia, činnosť Turingovho stroja sa zastaví (hovoríme, že Turingov stroj neakceptoval vstupné slovo).

1.4. POČÍTADLOVÝ STROJ

Počítadlový stroj (angl. a*bacus machine*, abacus = počítadlo) je jedným z najjednoduchších výpočtových modelov, ktorý je postavený na operáciách inkrementovania a dekrementovania.

Počítadlový stroj je definovaný ako reťazce nad istou abecedou. Abeceda pozostáva z nekonečného množstva symbolov $\mathbf{a_k}$ a $\mathbf{s_k}$, ľavých zátvoriek **(** a nekonečne veľa pravých zátvoriek $)_k$, kde index k je kladné celé číslo.

Jednoduchý počítadlový stroj s veľkosťou n je definovaný takto:

- ak a sk sú jednoduché počítadlové stroje s veľkosťou 0.
- počítadlové stroje s veľkosťou n sú reťazce M_1 ... M_r, kde každý reťazec M_i je jednoduchý počítadlový stroj s veľkosťou $\leq$ n a niektoré reťazce M_i majú veľkosť presne n.
- jednoduché počítadlové stroje s veľkosťou n+1 sú reťazce $(M)_k$, kde M je počítadlový stroj veľkosti n.

Rekurzívna definícia pozostáva z 3 častí:

- a_k a s_k sú počítadlové stroje, pričom $k \in N$,
- ak M_1, M_2, ... M_n sú počítadlové stroje, tak aj $M_1M_2...M_n$ je počítadlový stroj,
- ak M je počítadlový stroj, tak aj $(M)_k$ je počítadlový stroj, pričom $k \in N$,
  nič iné nie je počítadlový stroj.

Neformálny opis výpočtu počítadlového stroja

Počítadlový stroj pracuje s konečným počtom registrov R_i, kde index $i \in N$, počet registrov je zvolený podľa potreby. V registroch môžu byť uložené ľubovoľne veľké prirodzené čísla (nezáporné celé čísla). Sémantika jednotlivých častí rekurzívnej definície je nasledovná:

- Zápis $\mathbf{a_i}$ po sémantickej stránke znamená inkrementovanie registra R_i („a“ ako addition), čiže stroj a_i vykoná operáciu $R_i = R_i+1$, čím výpočet končí. Zápis $\mathbf{s_i}$ dekrementuje registrer Ri („s“ ako subtraction), ak bol tento nenulový. Čiže stroj s_i vykoná $R_i = R_i \ominus 1$ a jeho výpočet skončí. Operácia $x \ominus y$ je definovaná ako:

$$x \ominus y = \begin{cases} x - y & ak\ x \rangle y \\ 0 & inak \end{cases}$$

- Počítadlový stroj $M_1M_2...M_n$ vytvorený zreťazením počítadlových strojov M_1, $M_2...M_n$, vykoná operácie zodpovedajúce stroju M_1, potom stroju M_2 a takto ďalej až po stroj M_n, potom výčet končí.
- Zápisom $(M)_k$ sa realizuje cyklus. Najskôr sa otestuje register R_k, a ak je nenulový, vykoná sa operácia zodpovedajúca stroju M. Následne sa opätovne otestuje register

R_k, a ak je nenulový znovu vykoná stroj M. Stroj M sa cyklicky vykonáva, až kým register R_k nenadobudne nulovú hodnotu, potom výpočet zodpovedajúci stroju $(M)_k$ končí.

Na začiatku výpočtu sú všetky registre nastavené na nulu. Do zvolených registrov sa zapíšu vstupné údaje a počítadlový stroj začne vykonávať operácie tak, ako bolo uvedené v predchádzajúcej časti. Po skončení výpočtu je vo zvolených registroch výsledok výpočtu.

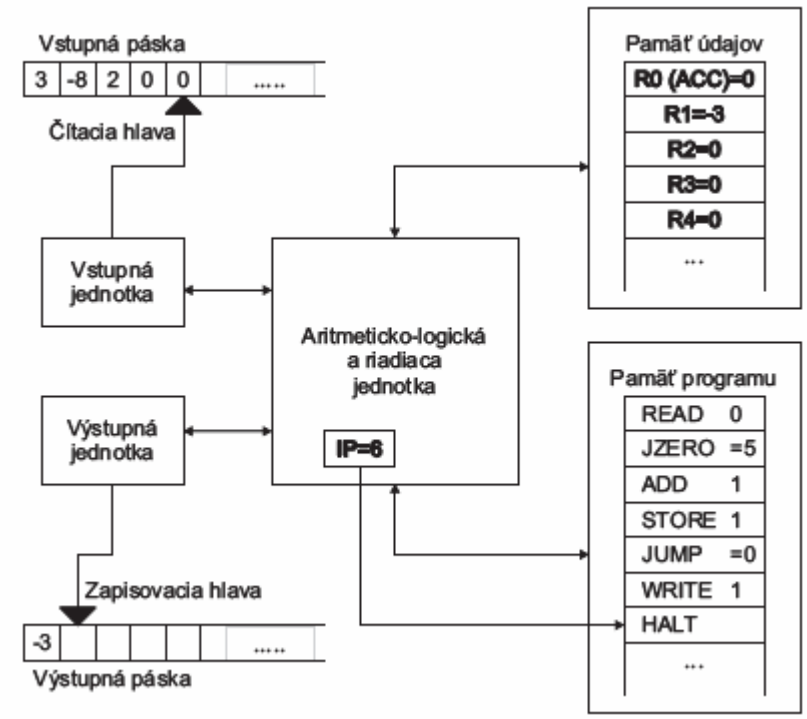
1.5. STROJ RAM

Stroj výpočtový RAM (angl. *Random Access Machine* - stroj s ľubovoľným prístupom do pamäti) je výpočtový model veľmi podobný klasickým sekvenčným počítačom. Bežný sekvenčný počítač pracuje deterministicky a jeho program je tvorený inštrukciami, ktoré väčšinou môžu pristupovať na ľubovoľné miesto v pamäti. RAM obsahuje pamäť dát tvorenú registrami a pamäť programu s inštrukciami. Vstupná jednotka a výstupná jednotka realizujú rozhranie s okolím a aritmeticko-logická a riadiaca jednotka interpretuje program a vykonáva operácie na základe inštrukcií.

Tieto vlastnosti predurčujú RAM ako vhodný výpočtový model na skúmanie algoritmov napr. z hľadiska výpočtovej či priestorovej zložitosti. Programovanie stroja RAM je podobné programovaniu jednoduchého procesora s veľmi obmedzenou sadou inštrukcií, a teda oboznámenie sa s činnosťou stroja môže slúžiť ako vhodný úvod do programovania v jazyku symbolických inštrukcií.

Neformálny opis stroja RAM

V pamäti údajov sú naznačené jednotlivé registre, ako aj ich obsah. V pamäti programu sú naznačené inštrukcie, adresy jednotlivých pamäťových buniek sú indexované od 0 (indexy nie sú označené). Taktiež sú na Obr. 5 naznačené obsahy vstupnej a výstupnej pásky. Načrtnutý program realizuje sčítanie poľa zo vstupnej pásky. Veľkosť poľa je určená ukončovacou hodnotou 0. Na obrázku Obr. 5 je naznačený stav stroja tesne pred ukončením výpočtu inštrukciou HALT, ktorú určuje ukazateľ aktuálnej inštrukcie (register IP).



Obr. 5: Schematické znázornenie jednotlivých blokov stroja RAM

Pamäť údajov

Pamäť údajov je tvorená registrami indexovanými od 0. Počet registrov je neobmedzený. Sú jediným miestom, kde je možné počas výpočtu udržiavať údaje. Do registra je možné uložiť celé číslo ľubovoľnej veľkosti (záporné či kladné celé číslo alebo nulu). Register je adresovaný inštrukciami určením jeho indexu. Nultý register, označený ako ACC, sa nazýva akumulátor a má špeciálny význam - je využívaný mnohými inštrukciami.

Pamäť programu

Základná, tu popísaná verzia stroja RAM má program oddelený od údajov, a taktiež nie je možná modifikácia programu za jeho behu. Program je tvorený jednotlivými inštrukciami vykonávanými sekvenčne. Inštrukcie sú usporiadané a na inštrukciu, ktorá sa vykoná v nasledujúcom kroku, ukazuje špeciálny register označený IP (instruction pointer). Po vykonaní aktuálnej inštrukcie sa register IP inkrementuje, a teda v ďalšom kroku bude vykonaná nasledujúca inštrukcia. Špeciálnym prípadom sú inštrukcie skokov, ktoré môžu tento register zmeniť, a tak riadiť vykonávanie programu. Inštrukcie sú popísané v časti „Inštrukčná sada“.

Vstupná jednotka

Úlohou vstupnej jednotky je zabezpečiť vstupné údaje pre výpočet. Údaje sú na políčkach vstupnej pásky a na aktuálne políčko ukazuje čítacia hlava. Po načítaní vstupu z aktuálneho políčka inštrukciou READ sa čítacia hlava posunie na nasledujúce políčko, a teda ďalšie vykonanie inštrukcie READ načíta vstup z ďalšieho políčka. Návrat čítacej hlavy na už prečítané políčka nie je možný. Vstupným údajom môže byť samozrejme údaj rovnakého typu a rozsahu ako aj v registri, a to je ľubovoľne veľké celé číslo.

Výstupná jednotka

Výstupná jednotka zabezpečuje výstup výsledkov výpočtu a pracuje podobne ako vstupná jednotka. Inštrukcia WRITE zapíše údaj na aktuálne políčko výstupnej pásky, určené zapisovacou hlavou, a zapisovacia hlava sa posunie na nasledujúce políčko. Návrat zapisovacej hlavy nie je možný, a teda už zapísané údaje nemôžu byť prepísané.

Aritmeticko-logická a riadiaca jednotka

Táto časť stroja riadi výpočet v závislosti na programe. Na základe registra ukazujúceho na aktuálnu inštrukciu (register IP) je inštrukcia náčítaná z pamäte programu a vykoná sa operácia podľa kódu a operandu inštrukcie. Riadiaca jednotka ovláda vstupnú aj výstupnú jednotku, a taktiež číta a zapisuje údaje z pamäte údajov tvorenej registrami. Po vykonaní operácie prislúchajúcej aktuálnej inštrukcii sa register IP zmení tak, aby ukazoval na inštrukciu, ktorá bude vykonaná v nasledujúcom kroku. Riadiace inštrukcie môžu register IP priamo meniť. Ostatné inštrukcie ho nemenia a register je inkrementovaný riadiacou jednotkou.

Inštrukčná sada a typy operandov

Inštrukčná sada stroja RAM je veľmi jednoduchá a obmedzená. Niektoré z inštrukcií musia mať uvedený práve jeden operand, ktorý určuje údaj, s ktorým inštrukcia bude pracovať'. Stroj RAM pracuje s troma typmi operandov uvedených v **Tab. 1**. Ako príklad použitia operandu je uvedená inštrukcia sčítania ADD, ktorá k obsahu akumulátora pričíta hodnotu podľa operandu a výsledok uloží opäť do akumulátora.

Operand	Príklad	Popis
= i	ADD = 5	Konštanta i. V príklade použitia sa k akumulátoru pripočíta číslo 5.
i	ADD 4	Priame adresovanie. Operand sa vyhodnotí ako obsah registra i. V príklade použitia sa k akumulátoru pripočíta obsah registra s indexom 4.
*i	ADD *7	Nepriame adresovanie. Obsah registra i určí register, ktorého obsah je výsledkom vyhodnotenia operandu. V príklade použitia sa k akumulátoru pripočíta obsah registra s indexom daným obsahom registra číslo 7.

Tab. 1: Typy operandov inštrukcií stroja RAM

Formálne je možné zaviesť operátor c(x), ktorý vráti hodnotu zodpovedajúcu obsahu registra s indexom x. Potom operátor v(op), ktorý vyhodnotí operand op, je možné definovať ako:

- v(=i) = i
- v(i) = c(i)
- v(*i) = c(c(i))

V prípade, ak by pri vyhodnotení operandu nastala nezmyselná situácia, a to pokus o čítanie registra so záporným indexom, sa výpočet stroja RAM zastaví.

Inštrukcia		
Kód	**Operand**	**Popis**
Inštrukcie pre prácu s pamäťou		
LOAD	operand	operand sa načíta do akumulátora
STORE	operand	obsah akumulátora sa zapíše do pamäte podľa operandu
Aritmetické inštrukcie		
ADD	operand	operand sa pripočíta do akumulátora (angl. *Addition*)
SUB	operand	operand sa odčíta od akumulátora (angl. *Substraction*)
MULT	operand	akumulátor sa prenásobí operandom (angl. *Multiplication*)
DIV	operand	akumulátor sa predelí operandom (angl. *Division*)
Vstupno výstupné inštrukcie		
READ	operand	údaj zo vstupnej pásky sa zapíše do pamäte podľa operandu
WRITE	operand	operand sa zapíše na výstupnú pásku
Riadiace inštrukcie		
JUMP	návestie	skok na návestie

JZERO	návestie	skok na návestie, ak akumulátor je nulový (angl. *Jump if zero*)
JGZERO	návestie	skok, ak akum. väčší ako nula (angl. *Jump if greater than zero*)
HALT		zastavenie výpočtu
ACCEPT		akceptovanie vstupu
REJECT		odmietnutie vstupu

Tab. 2: Inštrukčná sada stroja RAM

Výpočet na stroji RAM

Na začiatku výpočtu je pamäť programu prázdna a tiež sú všetky registre vynulované. Čítacia a aj zapisovacia hlava sú na začiatkoch vstupnej a výstupnej pásky a na vstupnú pásku sú umiestnené vstupné údaje. Program je zavedený do pamäte programu a register IP je nastavený na prvú inštrukciu (kedže je prvá inštrukcia umiestnená v pamäti programu na pamäťovom mieste s indexom 0, bol aj register IP nastavený na 0). Následne začne riadiaca jednotka vykonávať program. Vykonávanie programu je ukončené inštrukciou HALT, alebo ak nastane niektorá z nezmyselných situácií, ako napríklad pokus o čítanie či zápis do registra so záporným indexom alebo pokus o vykonanie inštrukcie z pamäťového miesta, na ktoré žiadna inštrukcia nebola umiestnená (neuvedenie inštrukcie HALT na konci programu). Výstup z programu je umiestnený na políčkach výstupnej páske až po políčko pred pozíciu zapisovacej hlavy.

1.6. PREHĽAD EXISTUJÚCICH RIEŠENÍ

Táto časť práce obsahuje analýzu existujúcich podobných riešení. Táto analýza nám poskytla cenné informácie, ktoré využijeme pri tvorbe nového simulátora. Taktiež nám umožní vytvoriť simulátor, ktorý využije dobré vlastnosti existujúcich simulátorov a tým sa z neho stane univerzálny nástroj a to nielen pre študentov, ale aj učiteľov zaoberajúcich sa touto problematikou.

V jednotlivých podkapitolách uvádzame krátky opis jednotlivých riešení, taktiež výhody a nevýhody, resp. to čo sa nám na danom simulátore páčilo a čo naopak nie. Pre lepšiu orientáciu a ohodnotenie kvality simulátora sme zaviedli stupnicu od 1 do 5, pričom 1 je najlepšia a 5 najhoršia známka. Obrázky sú uvedené iba pri vybraných dobrých riešeniach.

1.6.1. JFLAP

Popis

JFlap predstavuje samostatnú pokročilú Java aplikáciu na simuláciu FA, PDA, TM s rôznymi možnosťou vykonávania rozličných operácií. Umožňuje vstup zo súboru ako aj z grafického rozhrania. Poskytuje editor vstupnej pásky, ktorá je viditeľná počas simulácie ako aj čítaciu hlavu, ktorá simuluje pohyb po páske. Podporuje editáciu počiatočných a koncových stavov pri grafickom vstupe, deteminizimus, nedeteminizimus a simuláciu jednotlivých krokov. Po ukončení simulácie zobrazí kompletný výpočet avšak iba pre akceptujúce výpočty. Pri nedeterminizme zväčša uhádne správny krok.

Výhody

Pomerne jednoduché intuitívne ovládanie, možnosť zvýrazniť nedeterministické stavy a prechody.

Nevýhody

Poskytované operácie nie vždy dávajú korektné výsledky, mierne chaotické zobrazovanie prechodu na viac rôznych znakov.

Celkové hodnotenie: 1

1.6.2. VISUAL AUTOMATA SIMULATOR

Popis

Graficky simulátor, samostatná Java aplikácia umožňujúca simuláciu FA a TM. Umožňuje grafický vstup, textový vstup ako aj vstup zo súboru. Po zapnutí v menu sú viditeľné aj prechodové funkcie. Vstupná páska je viditeľná počas simulácie. Podporuje editáciu počiatočných a koncových stavov, deteminizimus, nedeteminizimus a grafický výstup Po ukončení simulácie zobrazí kompletný výpočet avšak iba pre akceptujúce výpočty. Pri nedeterminizme zväčša uhádne správny krok.

Výhody

Pekné grafické spracovanie.

Nevýhody

Nie je zobrazená simulácia, iba výsledok, iba pri debug móde. Taktiež nesprávne konvertuje NFA na DFA.

Celkové hodnotenie: 2

1.6.3. XTURINGMACHINELAB

Popis

Jednoduchý a prehľadný Java applet, ponúka aj hotové ukážky TM. Umožňuje textový vstup ako aj vstup zo súboru. Poskytuje editor vstupnej pásky, ktorá je viditeľná počas simulácie ako aj čítaciu hlavu, ktorá simuluje pohyb po páske. Podporuje editor prechodovej funkcie, deteminizimus, grafický výstup a simuláciu jednotlivých krokov. Po ukončení simulácie zobrazí kompletný výpočet avšak iba pre akceptujúce výpočty. Pri nedeterminizme zväčša uhádne správny krok. Pri tvorbe nového simulátora využijeme pohyb hlavy a časť používateľské rozhranie.

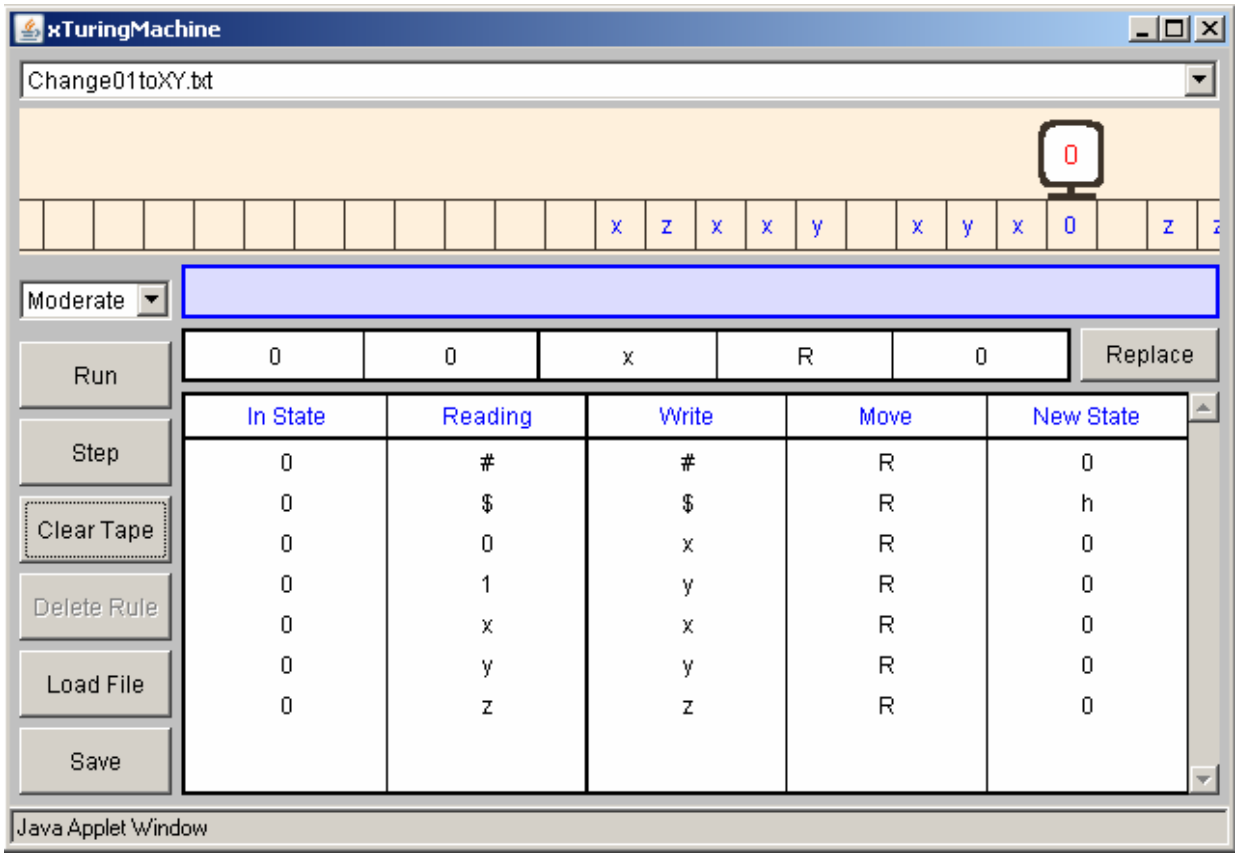
Výhody

Veľmi pekné používateľské rozhranie, hlavne znázornenie pohybu hlavy a pásky.

Nevýhody

Vstup vo formáte TXT.

Celkové hodnotenie: 2



Obr. 6: xTuringMachine

1.6.4. VISUAL TURING

Popis

C++ vizuálny návrh TM. Umožňuje vstup zo súboru ako aj grafický vstup . Na obrazovke je znázornený formálny zápis. Poskytuje editor vstupnej pásky, ktorá je viditeľná počas simulácia ako aj čítacia hlava, ktorá simuluje pohyb po páske. Podporuje editor prechodovej funkcie, editovanie počiatočných a koncových stavov, deteminizimus, grafický výstup, simuláciu jednotlivých krokov a nastavenie rýchlosti simulácie.

Výhody

Výborné vizuálne spracovanie, možnosť vloženia breakpointov a debuggovania aplikácie.

Nevýhody

Na prvý pohľad zložité ovládanie, nutnosť použitia manuálu – neintuitívne.

Celkové hodnotenie: 1

1.6.5. TURING MACHINE SIMULATOR

Popis

Jednoduchý C# program na simuláciu TM, ktorý je realizovaný ako Windows aplikácia. Umožňuje textový vstup ako aj vstup zo súboru. Poskytuje editor vstupnej pásky, ktorá je viditeľná počas simulácie ako aj čítacia hlava, ktorá simuluje pohyb po páske. Podporuje editor prechodovej funkcie, editáciu počiatočných a koncových stavov, deteminizimus, grafický výstup, simuláciu jednotlivých krokov a nastavenie rýchlosti simulácie.

Výhody

Nastavenie rýchlosti simulácie, voľba okamžitého výpočtu, načítanie/ukladanie súborov.

Nevýhody

Prechodové funkcie sa vkladajú pomerne neprirodzene a v programe sa označujú ako stavy. Možnosť používať na páske iba symboly 0 a 1, nepodporuje nedeterministický automat.

Celkové hodnotenie: 3

1.6.6. FSA APPLET

Popis

Veľmi jednoduchý applet pre simuláciu FA s vizualizáciou stavového diagramu. Umožňuje vstup zo súboru ako aj grafický vstup. Poskytuje editor vstupnej pásky, ktorá je viditeľná počas simulácie ako aj čítacia hlava, ktorá simuluje pohyb po páske. Podporuje editáciu počiatočných a koncových stavov, deteminizimus, grafický výstup a simuláciu jednotlivých krokov.

Výhody

Pomerne jednoduché intuitívne ovládanie s krátkou nápovedou.

Nevýhody

Pri vytvorení nedeterministického automatu sa snaží simulovať nedeterminizmus, ale nekorektne zamietne správny vstup.

Celkové hodnotenie: 2

1.6.7. SIMULÁTOR TURINGOVA STROJE (STEHLÍK)

Popis

Jednoduchá aplikácia používaná na cvičeniach z predmetu TZI na simulovanie TM. Umožňuje textový vstup ako aj vstup zo súboru. Na obrazovke je znázornený formálny zápis. Poskytuje editor vstupnej pásky, ktorá je viditeľná počas simulácie ako aj čítaciu hlavu, ktorá simuluje pohyb po páske. Podporuje možnosť meniť vstupnú pásku počas behu programu, editáciu počiatočných a koncových stavov, deteminizimus, nedeteminizimus, simuláciu jednotlivých krokov. Poskytuje podsvietenie syntaxe ako aj vyznačenie chýb syntaxe. Pri nedeterminizme zväčša uhádne správny krok.

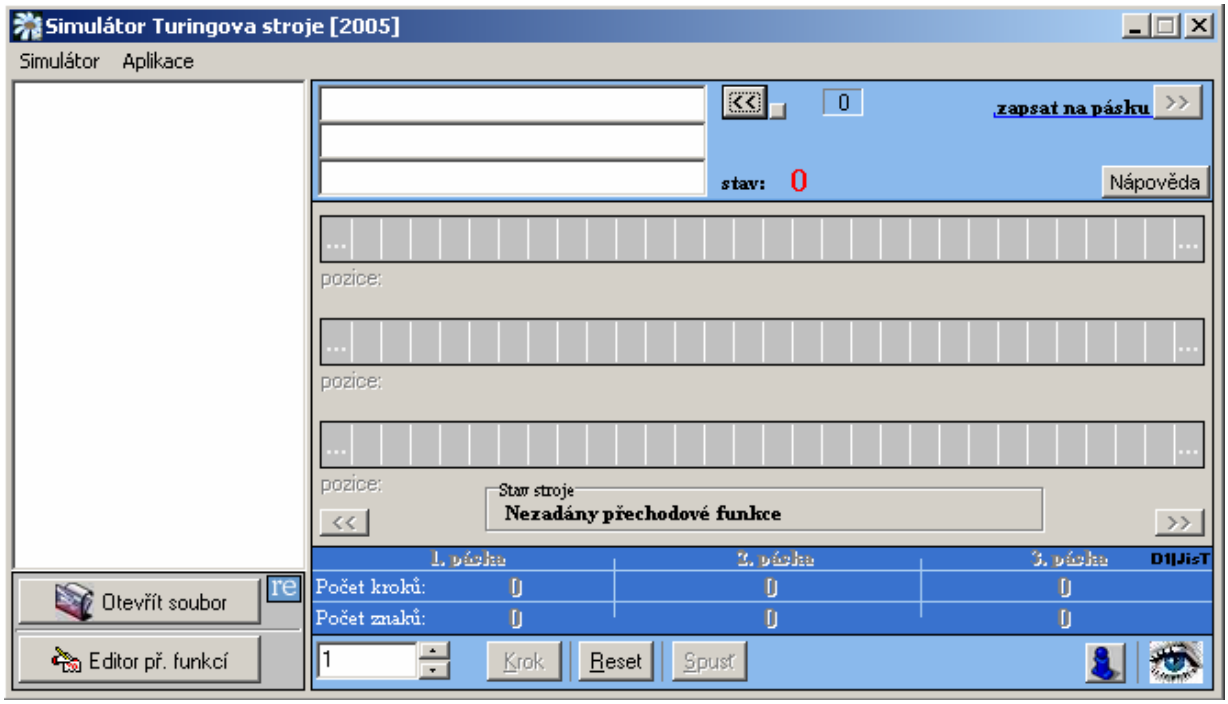
Výhody

Jednoduché intuitívne ovládanie. Všetky znaky sa dajú uložiť na pásku. Komentáre, pekne simuluje výpočet po krokoch. Vlastné vstupy a práca so súborom.

Nevýhody

Chýba grafická interpretácia. Pri nedeterminizme vyberie len prvú možnosť, o ostatné sa nezaujíma.

Celkové hodnotenie: 2



Obr. 7: Simulátor Turingova Stroje (Stehlík)

1.6.8. ŠTUDENTSKÁ PRÁCA – PAŠMÍK

Popis

Samostatná Java aplikácia na FA a PDA. Umožňuje grafický vstup, textový vstup ako aj vstup zo súboru. Na obrazovke je znázornený formálny zápis. Poskytuje editor vstupnej pásky, čítacia hlava je viditeľná a simuluje pohyb po páske. Podporuje editor prechodovej funkcie, editovanie počiatočných a koncových stavov, deteminizimus, nedeteminizimus, grafický výstup, simuláciu jednotlivých krokov a nastavenie rýchlosti simulácie. Zobrazuje kompletný výpočet ako aj strom všetkých možností. Poskytuje možnosť voliť nasledujúci krok výpočtu pri nedeterminizme, pri ktorom zväčša uhádne správny krok.

Výhody

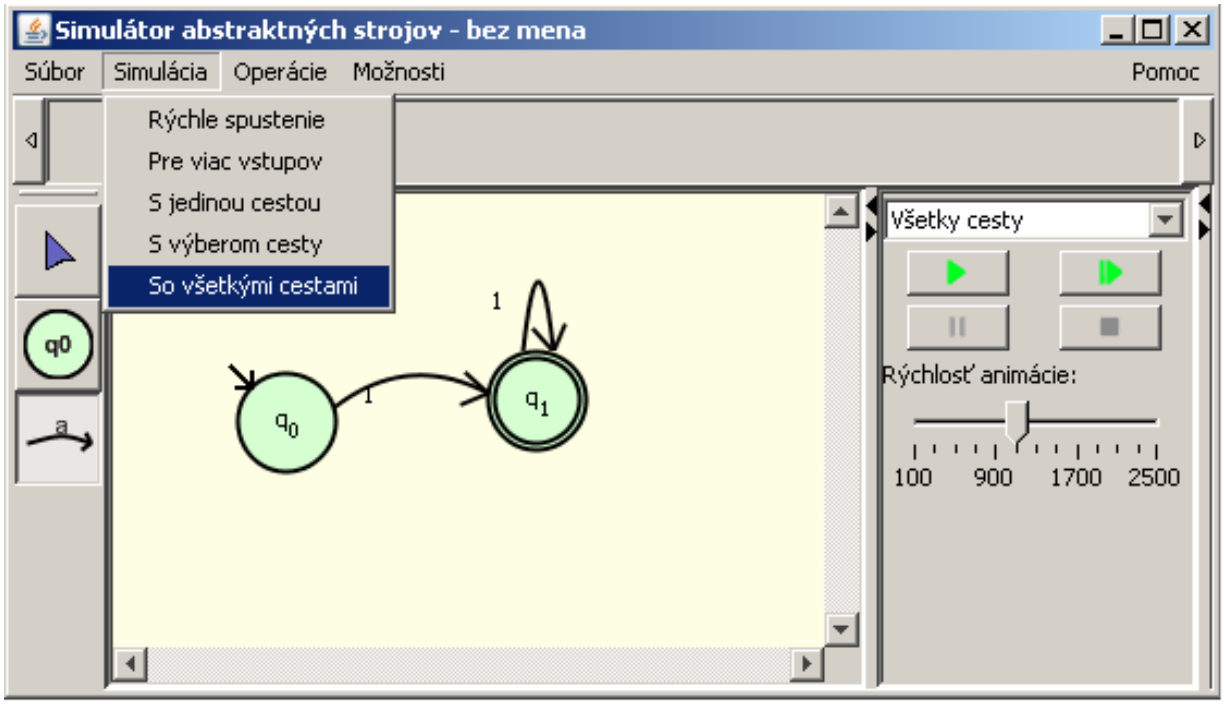
Pekné používateľské rozhranie, jednoduché na používanie. Nastavenie rýchlosti simulácie a import/export do JFLAP.

Nevýhody

Nedoladené niektoré časti používateľského rozhrania – napr. neprimeraná veľkosť okien, nefunguje pohyb pomocou šípiek... Nesprávne transformuje deterministický automat, ak

zvolíme „konverziu NKA na DKA“, urobí automat zbytočne zložitý (netestuje či už nie je deterministický).

Celkové hodnotenie: 2



Obr. 8: Študentská práca – Pašmik

1.6.9. ŠTUDENTSKÁ PRÁCA – KOHAUT

Popis

Tri samostatné aplikácie na simuláciu FA, PDA, TM. Umožňuje vstup zo súboru. Poskytuje editor vstupnej pásky a čítacia hlava je viditeľná a simuluje pohyb po páske. Podporuje deteminizimus, grafický výstup a simuláciu jednotlivých krokov. Poskytuje vyznačenie chýb syntaxe.

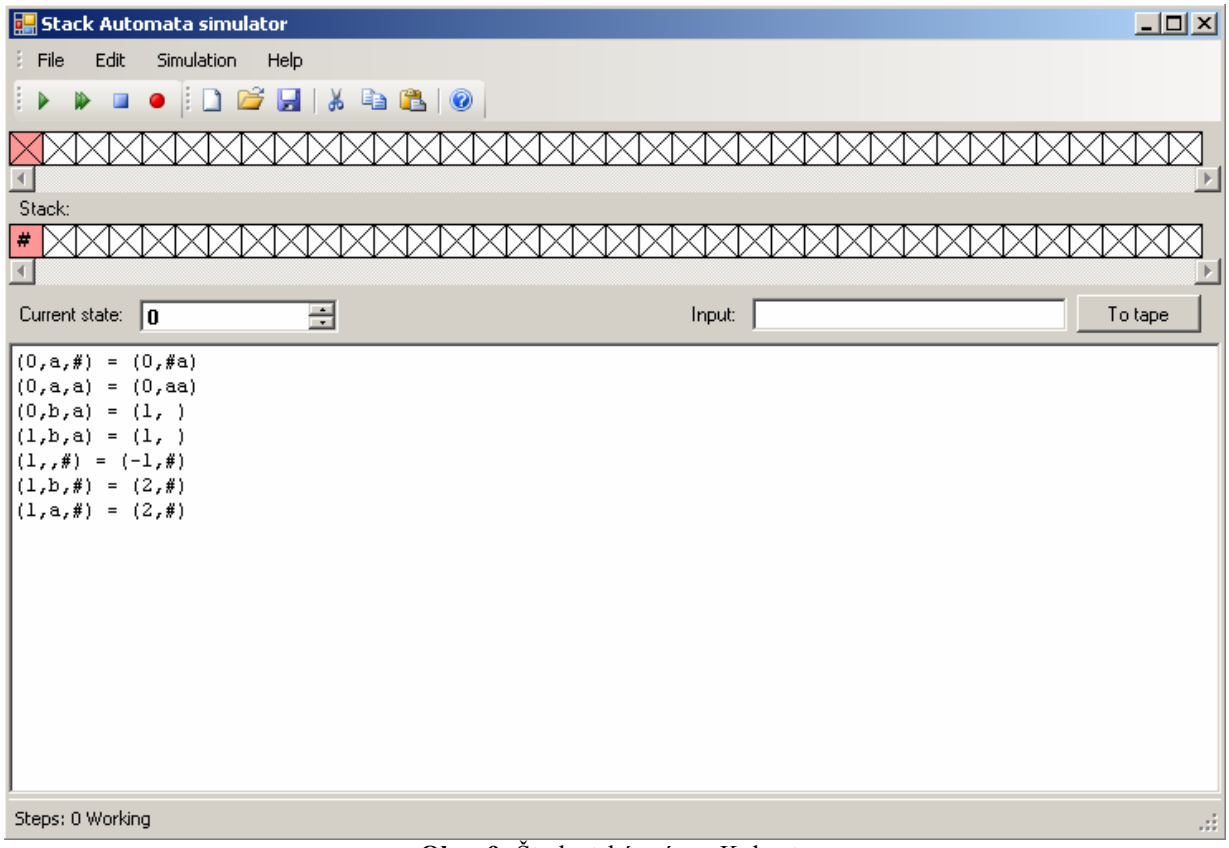
Výhody

Pekne grafické rozhranie. Na základe funkcie vie vygenerovať obrázok.

Nevýhody

Ťažkopádne vstupy. Podporuje načítanie iba z obyčajného textového súboru.

Celkové hodnotenie: 3



Obr. 9: Študentská práca - Kohaut

1.6.10. ŠTUDENTSKÁ PRÁCA – PORUBČAN

Popis

Tri samostatné Java aplikácie pre simuláciu FA, PDA a TM. Umožňuje textový vstup ako aj vstup zo súboru. Na obrazovke je znázornený formálny zápis. Poskytuje editor vstupnej pásky, ktorá je viditeľná počas simulácie ako aj čítacia hlava, ktorá simuluje pohyb po páske. Podporuje deteminizimus, nedeteminizimus a simuláciu jednotlivých krokov.

Výhody

Pomerne jednoduché intuitívne ovládanie s krátkou nápovedou.

Nevýhody

Chýba grafický výstup.

Celkové hodnotenie: 3

1.6.11. ŠTUDENTSKÁ PRÁCA – KULIHA

Popis

Java aplikácia na simuláciu FA a PDA. Umožňuje grafický vstup, textový vstup ako aj vstup zo súboru. Na obrazovke je znázornený formálny zápis. Poskytuje editor vstupnej pásky. Podporuje editor prechodovej funkcie, editovanie počiatočných a koncových stavov, nedeteminizimus, grafický výstup, simuláciu jednotlivých krokov.

Výhody

Pekný graficky editor, príjemné používateľské rozhranie.

Nevýhody

Pomalý výpočet pri nedeterminizme a chyby pri simulácii.

Celkové hodnotenie: 2

1.6.12. ŠTUDENTSKÁ PRÁCA – RODINA

Popis

Samostatná .Net aplikácia na TM, RAM a AM. Umožňuje grafický vstup, textový vstup ako aj vstup zo súboru. Na obrazovke je znázornený formálny zápis. Poskytuje editor vstupnej pásky a čítacia hlava je viditeľná a simuluje pohyb po páske. Podporuje editor prechodovej funkcie, editáciu počiatočných a koncových stavov, deteminizimus, nedeteminizimus, grafický výstup, simuláciu jednotlivých krokov, nastavenie rýchlosti simulácie a podsvietenie syntaxe. Zobrazuje kompletný výpočet ako aj strom všetkých možností. Pri nedeterminizme zväčša uhádne správny krok.

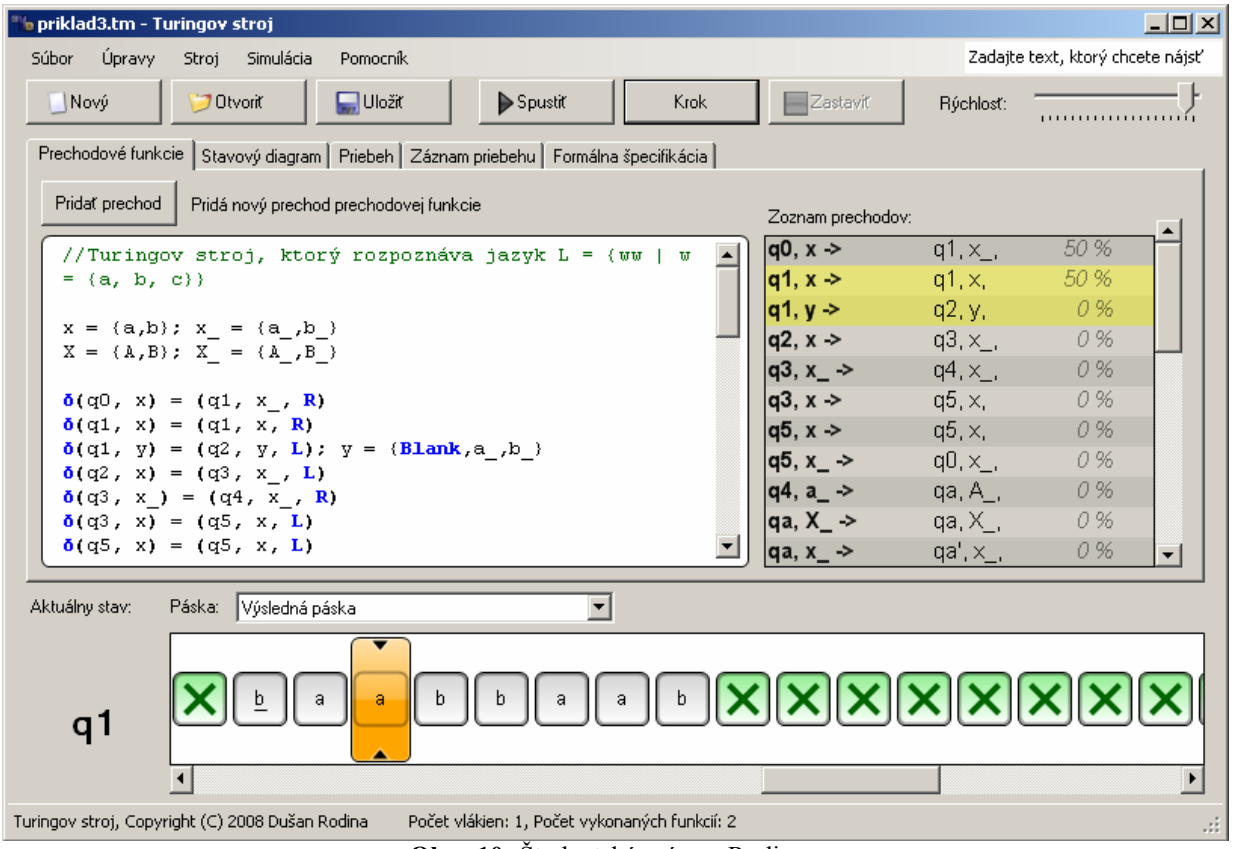
Výhody

Pekné používateľské rozhranie, jednoduché na používanie. Simulácia hlavy a pásky. Nastavenie rýchlosti simulácie.

Nevýhody

Nemožnosť editovať v grafickom móde.

Celkové hodnotenie: 1



Obr. 10: Študentská práca – Rodina

1.6.13. ABACUSMACHINESIMULATOR – DUŠAN RODINA

Popis

Prehľadne a príjemne spracovaný počítačový stroj. Ako pozitívum hodnotím oddelenie samostatnej záložky na simuláciu, kde sú názorne vykreslené jednotlivé kroky simulácie programu.

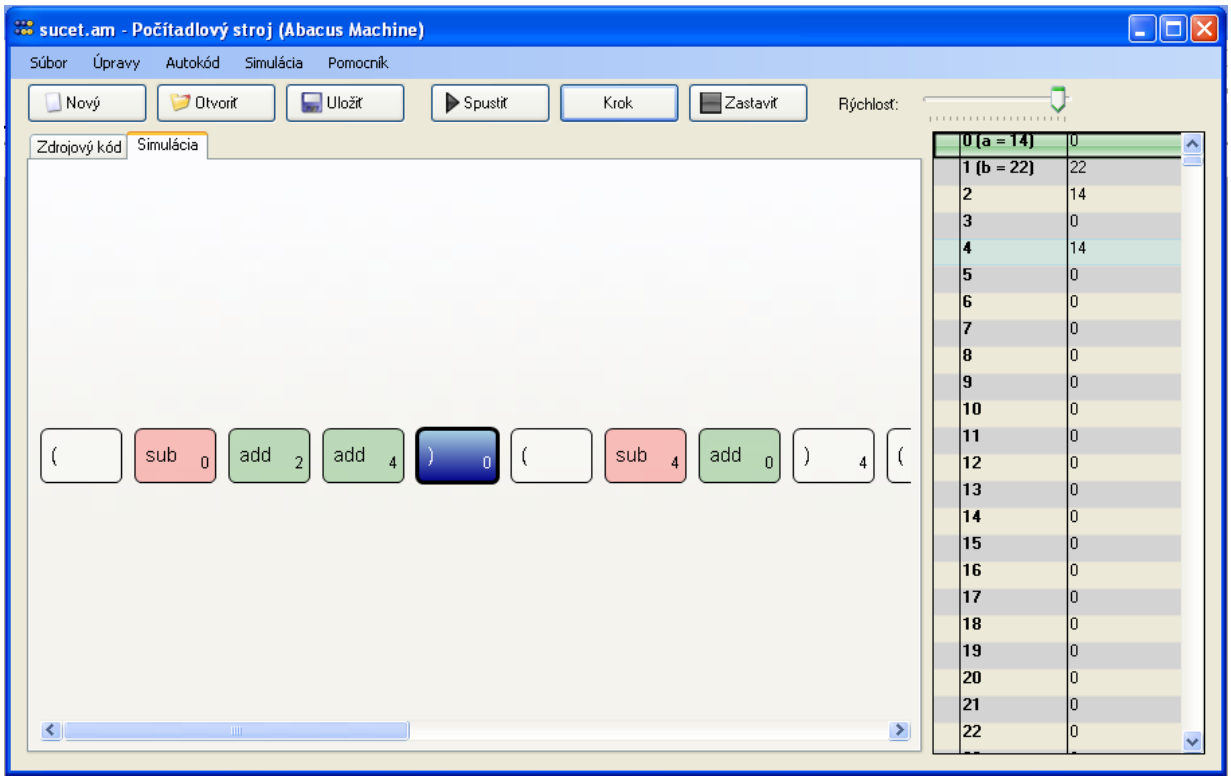
Ďalším pozitívom je predprogramovanie základných úkonov ako kopírovať alebo presunúť hodnotu z jedného registra do druhého, vynulovať register.

Výhody

Simulácia zmeny hodnoty registra a presúvanie hodnôt medzi registrami.

Nevýhody

Celkové hodnotenie: 1



Obr. 11: AbacusMachineSimulator – Dušan Rodina

1.6.14. ROD REGO (REGISTER MACHINE SIMULATION) - DANIEL DENNETT

Popis

V tomto simulatore sú čísla v registroch zobrazované pmocou počtu guličiek. Jednotlivé inštrukcie sú očíslované. Simulator umožňuje registre inkrementovať, dekrementovať a vytvoriť cykly. Inštrukcie pozistávaju zo skratky príkazu inc pre inkrementovanie a deb pre dekrementovanie, nasleduje číslo registra, ďalej číslo inštrukcie ktorá sa vykonáva kým nie je register nulový a posledná číslica označuje inštrukciu, ktorou má program po vykonaní cyklu pokračovať.

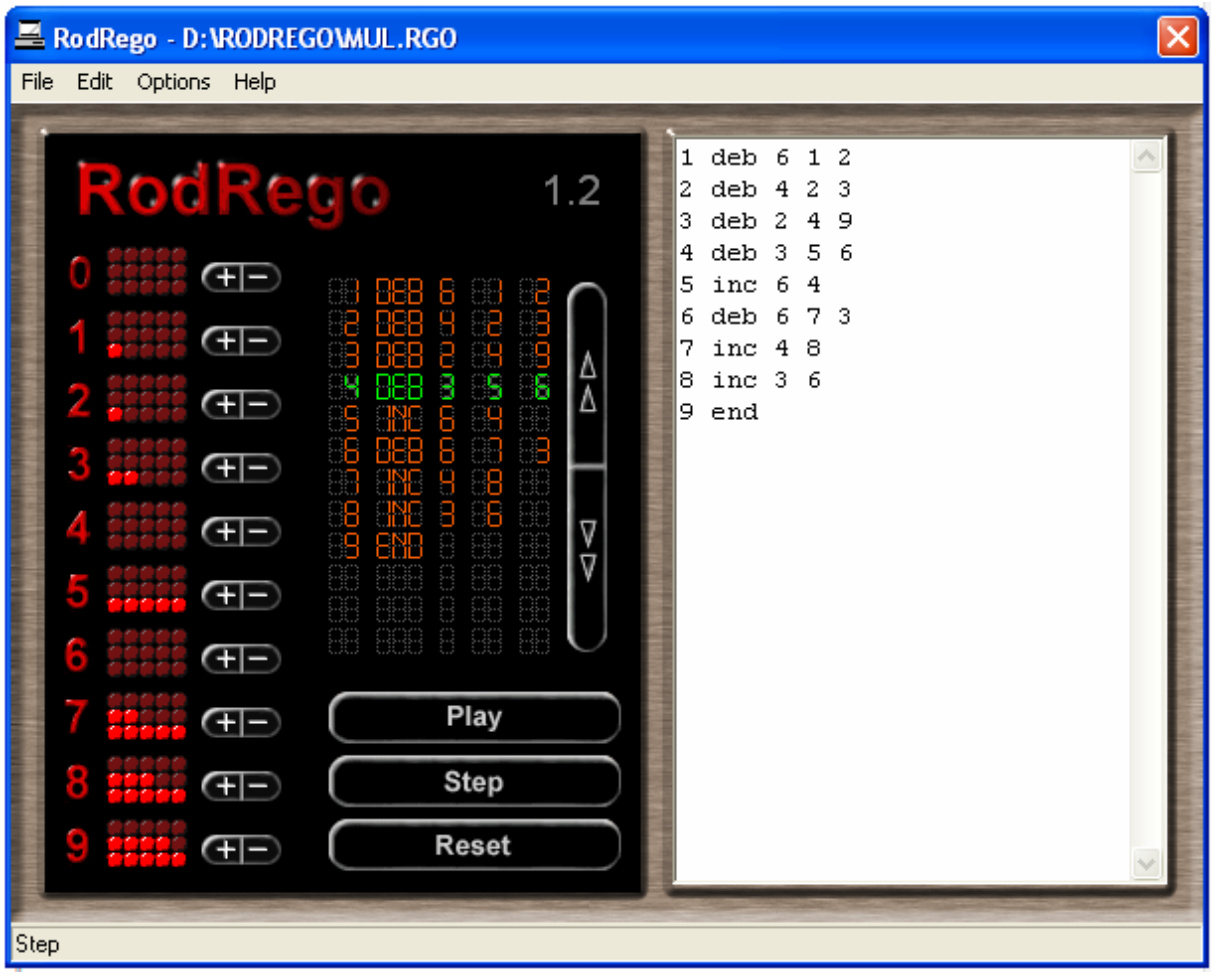
Výhody

Myšlienka z dema program využiť stavový diagram na simuláciu.

Nevýhody

Ťažké sledovanie postupnosti v simulácii, kam skáču jednotlivé inštrukcie.

Celkové hodnotenie: 2



Obr. 12: Simulátor RodRego

1.6.15. REGISTER MACHINES

Popis

V tomto prípade je nutnosť inštalácie Mathematica Player 7, ktorý je free. Tento simulátor ovláda len dva druhy inštrukcií a to inkrementovanie a dekrementovanie. Možnosť navoliť si počet registrov.

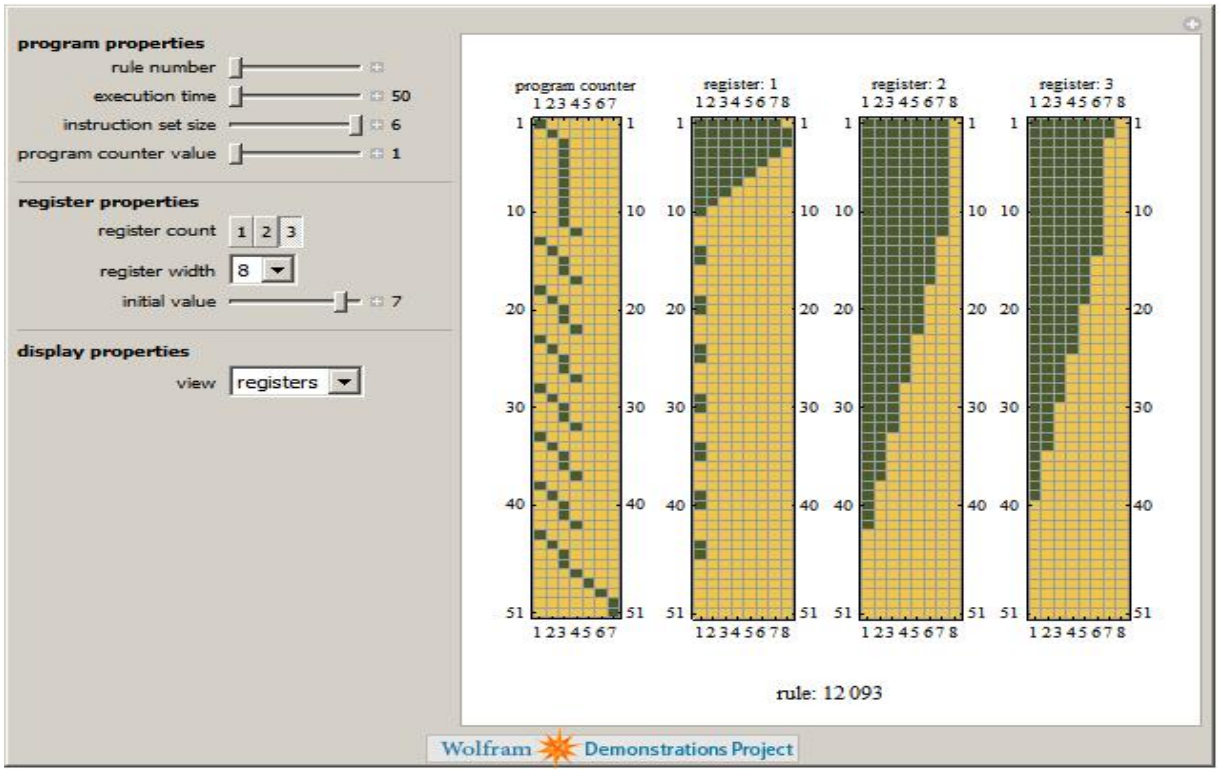
Výhody

Možnosť nastavenia počtu registrov.

Nevýhody

Nutnosť prepínania medzi programom a samotnými registrami. Neovláda inštrukciu cyklu.

Celkové hodnotenie: 4



Obr. 13: RegisterMachines

1.6.16. SIM STUDIO – DUŠAN RODINA

Popis

Aplikácia umožňujúca prácu s uloženými programami, aj tvorbu vlastných programov. Umožňuje vloženie zdrojového kódu z jazyka C. Simulácia pekne graficky spracovaná s možnosťou voľby rýchlosti simulácie, umožňuje aj krokovanie. Podrobne zobrazuje prácu simulátora. Počíta aj zložitosť programu.

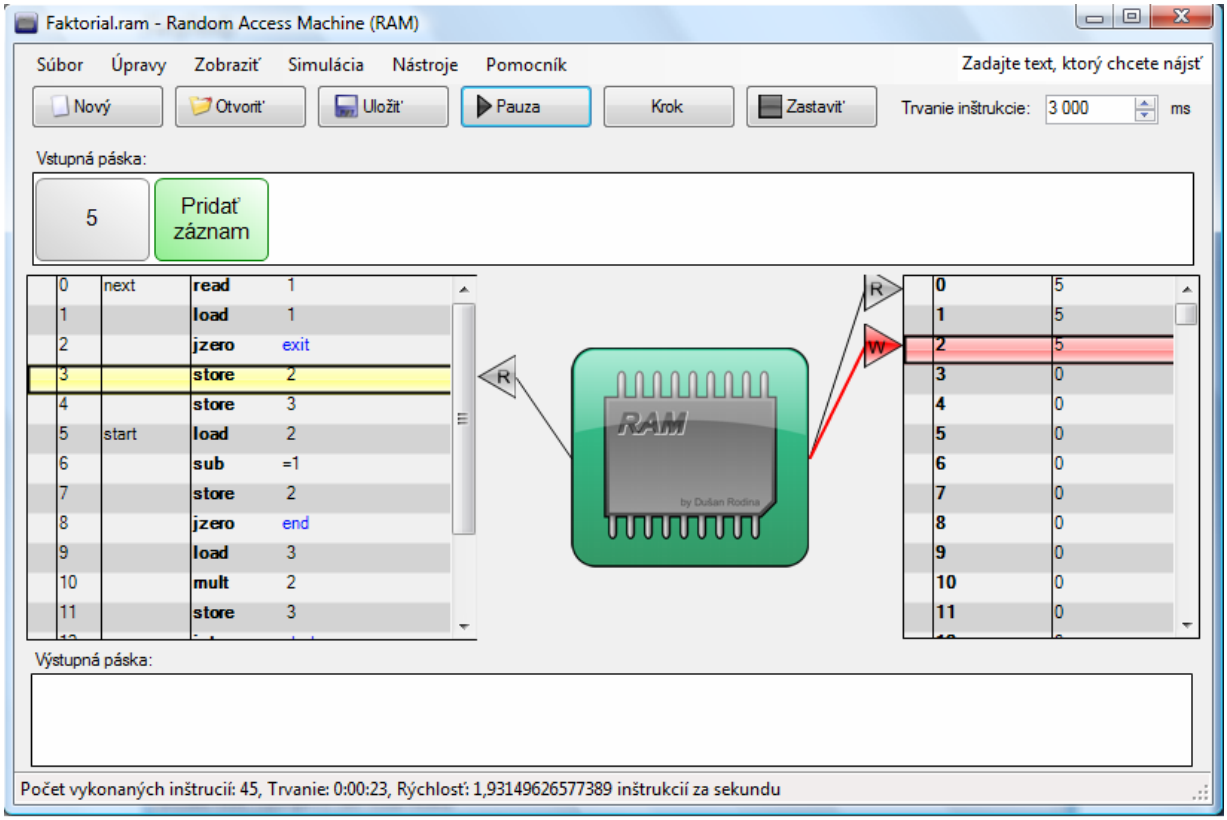
Výhody

Jednoduché ovládanie, jasne a prehľadne zobrazená simulácia. Spracuje naraz viac vstupov.

Nevýhody

Chýba kontrola, či vstup je korektný. Bola by vhodná aj možnosť spätného kroku.

Celkové hodnotenie: 1



Obr. 14: Simulácia RAM simulátora - Dušan Rodina

1.6.17. RAM MACHINE 2006 (ŁUKASZ SZKUP)

Popis

Aplikácia, ktorá jednoduchou simuláciou predvádza činnosť simulátora. Pracuje so súbormi, umožňuje jednoduché tvorenine vlastných programov – ponúka možnosti inštrukcií. Obsahuje vstupnú a výstupnú pásku. Dajú sa meniť rôzne nadstavenia.

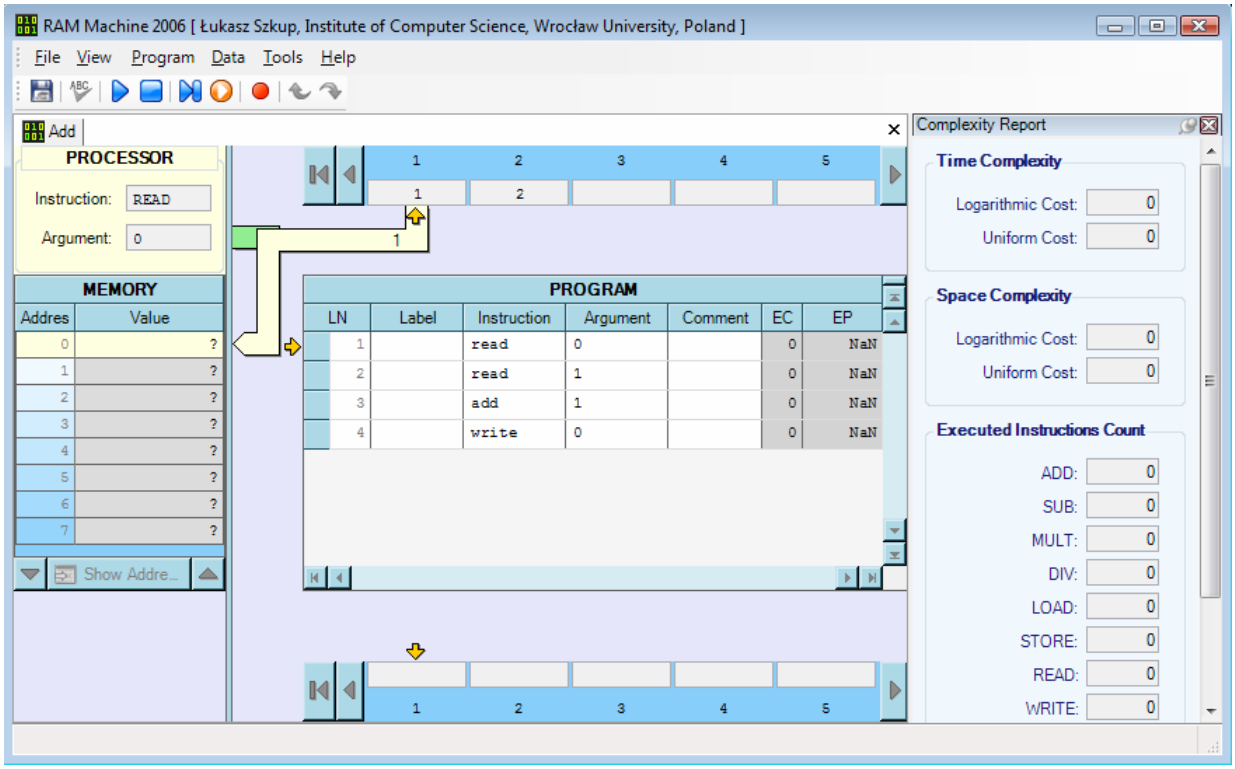
Výhody

Umožňuje kontrolu správnosti programu (kompilátor), umožňuje aj prácu s breakpointami. Prepracovaná vizualizácia postupu vykonávania programu – zobrazuje tok inštrukcií medzi páskami, programom, procesorom a pamäťou. Ponúka možnosť zobrazenia štatistiky zložitosti programu.

Nevýhody

Zdrojový súbor je uložený ako text. Trocha horšie ovládanie rolovacieho menu.

Celkové hodnotenie: 1



Obr. 15: RAM Machine - Łukasz Szkup

1.6.18. JAVA RAM INTERPRETER 1.0 (JAM)

Popis

Java simulátor, ktorý umožňuje načítanie a editovanie zdrojových súborov a ich ukladanie. Poskytuje simuláciu v dvoch rýchlostiach, alebo po krokoch. Postup simulácie zobrazený v tabuľke.

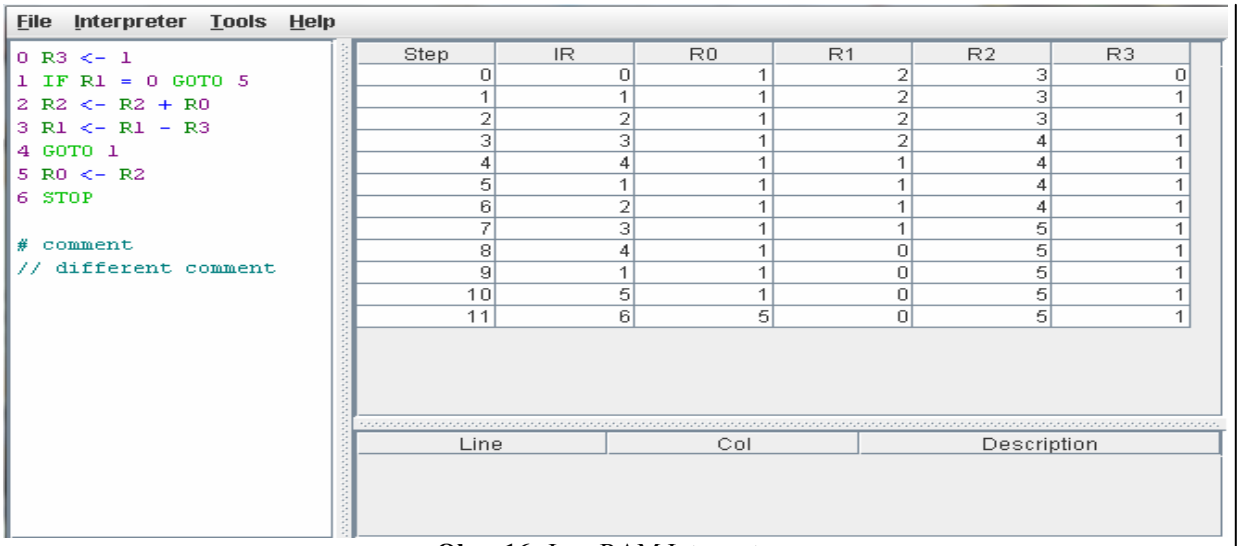
Výhody

Jednoduché ovládanie, obsahuje 2 jednoduché príklady, prehľadné zobrazenie postupu simulácie.

Nevýhody

Obmedzaním je potreba číslovania riadkov v kóde. Neobsahuje žiadne rýchle tlačítka, ku všetkému sa musí pristupovať cez menu. Nepoužíva klasickú syntax RAM simulátorov. Nie je zobrazená vstupná ani výstupná páska. Stiahnuteľný súbor obsahuje zdrojové súbory, ktoré je potrebné skompilovať.

Celkové hodnotenie: 4



Obr. 16: Java RAM Interpreter

1.6.19. RASP – PETER PRIKRYL

Popis

Jednoduchá aplikácia spustiteľná aj priamo v prehliadači. Jednotlivé kroky sú zvýraznené farebne.

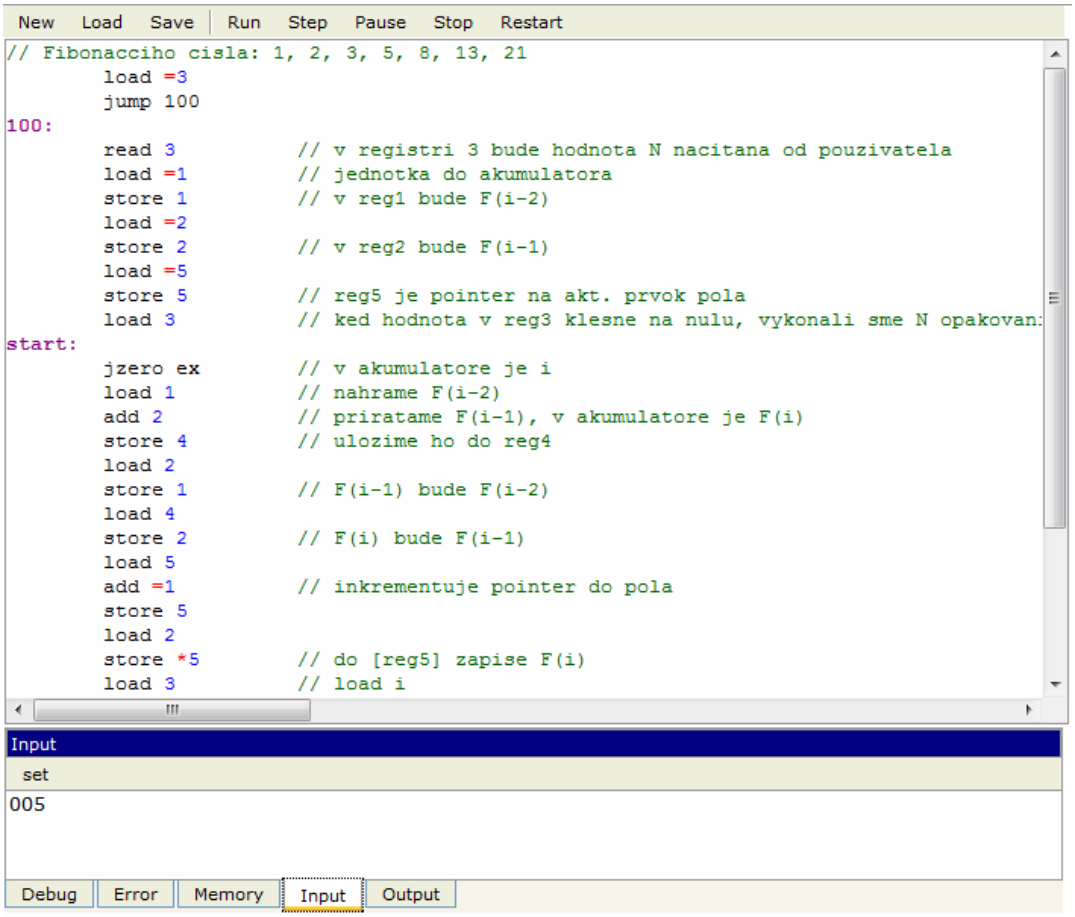
Výhody

Zdrojové súbory uložené vo formáte XML. Vhodný na rýchle otestovanie programu.

Nevýhody

Medzi jednotlivými oknami (vstup, výstup, pamäť, error a debug) je potrebné sa preklikávať, nie je možné vidieť ich naraz.

Celkové hodnotenie: 3



Obr. 17: RASP – Peter Prikryl

1.6.20. RAM SIMULATOR – RICHARD VESELÝ

Popis

Jednoduchá aplikácia, ktorá pracuje so súbormi, vstupnou a výstupnou páskou. Umožňuje vkladanie breakpointov. Pri vytváraní programu poskytuje návrhy inštrukcií. Podporuje krokovanie simulácie aj vykonanie programu v 1 kroku rôznou rýchlosťou.

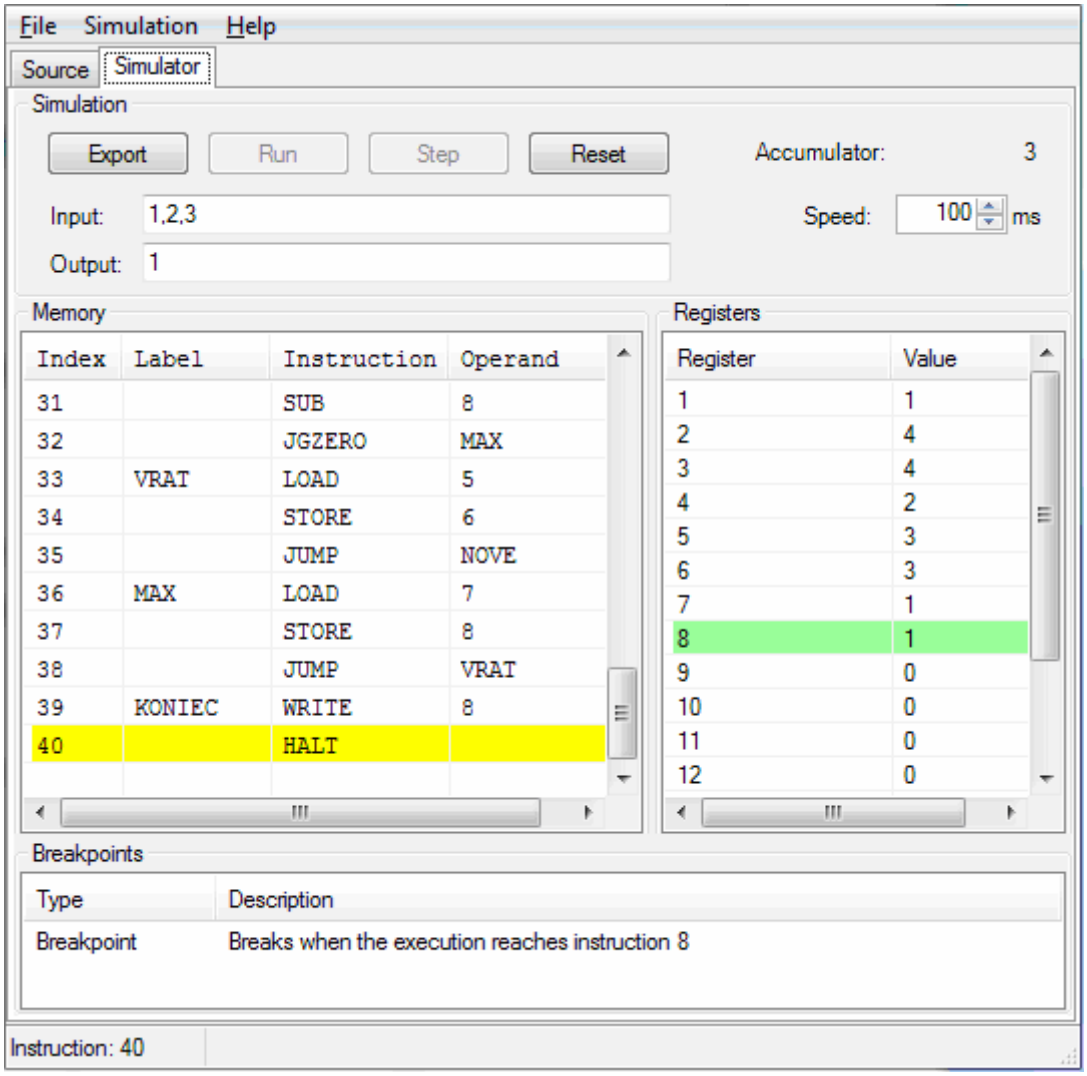
Výhody

Pri vstupnej páske zobrazuje formát vstupu. Program sa dá písať v ľubovoľnom textovom editore alebo priamo v programe.

Nevýhody

Zdrojový kód uložený vo formáte txt.

Celkové hodnotenie: 2



Obr. 18: RAM simulátor – Richard Veselý

1.7. METODIKA HODNOTENIA SIMULÁTOROV

Pri porovnávaní hotových riešení simulátorov sme hodnotili, aké funkčné vlastnosti majú a na akej úrovni spracovania sú. Najprv sme do pripravenej tabuľky doplnili údaje o konkrétnom simulátore pre identifikáciu a ohodnotili sme ho podľa nasledujúcich bodov:

1. Celkové subjektívne hodnotenie (1-5)
2. Na obrazovke je znázornený formálny zápis (množina stavov, vstupná abeceda, štartovací stav, finálne stavy, prechodová funkcia,....)
3. Súborový vstup
4. Textový vstup
5. Grafický vstup
6. Hlava viditeľná, simulácia pohybu

7. Vstupná páska viditeľná
8. Editor vstupnej pásky
9. Možnosť meniť vstupnú pásku počas behu "programu"
10. Editor pre štartovací a koncové stavy
11. Editor prechodovej funkcie
12. Podpora "wildcards" $(q_0,x)=q_1$, $x \in \{a,b,c\}$
13. Syntax highlight
14. Vyznačenie chýb syntaxe
15. Grafický výstup
16. Simulácia krokov (po krokoch/v celku)
17. Nastavenie rýchlosti simulácie
18. Podporuje deteminizimus
19. Zobrazenie kompletného celého výpočtu
20. Podporuje nedeteminizmus
21. Zobrazenie všetkých výpočtov – strom
22. Mmožnosť voliť nasledujúci krok výpočtu pri nedeterminizme
23. "Uhádne" správne riešenie pri nedeterminizme

Po vyplnení týchto údajov sme napísali vlastné postrehy – výhody, nevýhody, iné.

Nasledujúci krok bolo vzájomné porovnávanie, zisťovanie všetkých výhod a nevýhod, výber vlastností, ktoré by sme chceli aplikovať do nášho simulátora, a zhrnutie najdôležitejších vlastností do tabuľky.

Názov	Výhody	Nevýhody	Známka	Poznámka
JFlap	Jednoduché intuitívne ovládanie, možnosť zvýrazniť nedeterministické stavy a prechody.	Poskytované operácie nie vždy dávajú korektné výsledky, mierne chaotické zobrazovanie prechodu na viac rôznych znakov.	1	Robustný nástroj s mnohými funkciami patrí medzi tie, ktoré využijeme pri tvorbe nového simulátora.
Visual Automata Simulator	Pekné grafické spracovanie.	Nevidno simuláciu iba výsledok, iba pri debug móde. Taktiež zle konvertuje NFA na DFA.	2	
XTuringMachineLab	Veľmi pekné GUI, hlavne znázornenie pohybu hlavy a pásky.	Vstup vo formáte TXT.	2	Využijeme simuláciu pohybu hlavy po páske.
Visual Turing	Výborné vizuálne spracovanie, možnosť vloženia breakpointov a debuggovania aplikácie.	Na prvý pohľad zložité ovládanie, nutnosť manuálu – neintuitívne.	1	
Turing Machine Simulator	Nastavenie rýchlosti simulácie, voľba okamžitého výpočtu, načítanie/ukladanie súborov.	Neprirodzené vkladanie prechodových funkcií. Podporované symboly 0 a 1, nepodporuje nedeterminizmus.	3	Využijeme nastavenie rýchlosti simulácie a prácu so súbormi.
FSA Applet	Intuitívne ovládanie s krátkou nápovedou.	Pri nedeterminizme nekorektne zamietne správny vstup.	2	
Simulátor Turingova Stroje (Stehlík)	Intuitívne ovládanie. Všetky znaky sa dajú uložiť na pásku. Komentáre, pekne simuluje výpočet po krokoch. Vlastne vstupy a práca so súborom.	Chýba grafická interpretácia. Pri nedeterminizme vyberie len prvú možnosť.	2	Využijeme výpočet po krokoch.

Pašmik	Pekne GUI, jednoduché na používanie. Nastavenie rýchlosti simulácie a import/export do JFLAP.	Nedoladene niektoré časti GUI.	2	Využijeme nastavenie rýchlosti simulácie a import export do JFlap.
Kohaut	Pekne grafické rozhranie. Na základe funkcie vie vygenerovať obrázok.	Ťažkopádne vstupy. Podporuje načítanie iba z obyčajného textového súboru.	3	Využijeme vygenerovanie obrázku.
Porubčan	Pomerne jednoduché intuitívne ovládanie s krátkou nápovedou.	Chýba grafický výstup.	3	
Kuliha	Pekný graficky editor, prijemne GUI.	Pomaly výpočet pri nederminizme a chyby pri simulácii.	2	
Rodina	Pekne používateľské rozhranie, jednoduché na používanie. Simulácia hlavy a pásky. Nastavenie rýchlosti simulácie.	Nemožnosť editovať v grafickom móde.	1	Využijeme GUI, pohyb po páske, priebeh riešenia.
SIM Studio Dušan Rodina	Pracuje so súbormi, prepracovaná simulácia, pekné GUI, počítanie zložitosti výpočtu	Nekontroluje korektnosť vstupu.	1	Jednoduchý nástroj s peknou simuláciu.
RAM machine (Łukasz Szkup)	Pri písaní programu ponúka na výber inštrukcie, obsahuje kompilátor. Veľmi pekne prepracovaná simulácia toku inštrukcií.	Pracuje s textovými súbormi.	1	Pekne zobrazená simulácia.
Java RAM Interpreter 1.0	Jednoduché ovládanie	Nepoužíva klasickú syntax RAM simulátorov.	4	

RASP – Peter Prikryl	Jednoduchá aplikácia spustiteľná v prehliadači, pracuje s XML súbormi	Nie je možné vidieť viac okien (vstup, výstup, pamäť...) naraz.	3	
RAM Simulator – Richard Veselý	Pri vstupnej páske zobrazuje formát vstupu. Program sa dá písať v ľubovoľnom textovom editore alebo priamo v programe.	Zdrojový kód uložený vo formáte txt.	2	Využijeme, že pri vytváraní programu poskytuje návrhy inštrukcií.
Abacus machine simulator - Dušan Rodina	Simulácia zmeny hodnoty registra a presúvanie hodnôt medzi registrami.		1	Pekne zobrazená simulácia
Rod Rego	Hodnoty registrov sú ľahko sledovateľné počas simulácie.	Ťažké sledovanie postupnosti v simulácii, kam skáču jednotlivé inštrukcie.	2	Myšlienka z dema – využiť stavový diagram na simuláciu.
Register Machines	Možnosť nastavenia šírky registrov.	Nutnosťprepínania medzi programom a samotnými registrami. Neovláda inštrukciu cyklu.	4	

Tab. 3: Porovnanie simulátorov. Podpora ponúkaných funkcií.

Názov	**Simulátor**	**Funkcie**
JFlap	FA, PDA, TM	vstup zo súboru, grafický vstup, editor vstupnej pásky, páska viditeľná počas simulácia, čítacia hlava simuluje pohyb po páske, editácia počiatočných a koncových stavov, deteminizimus, nedeteminizimus, simulácia jednotlivých krokov
Visual Automata Simulator	FA, TM	grafický vstup, textový vstup, vstup zo súboru, viditeľné prechodové funkcie, viditeľná vstupná páska, editácia počiatočných a koncových stavov, deteminizimus, nedeteminizimus, grafický výstup

XTuringMachineLab	TM	textový vstup, vstup zo súboru, editor vstupnej pásky, páska viditeľná počas simulácie, čítacia hlava simuluje pohyb po páske, editor prechodovej funkcie, deteminizimus, nedeterminizmus, grafický výstup, simulácia jednotlivých krokov.
Visual Turing	TM	vstup zo súboru, grafický vstup, na obrazovke znázornený formálny zápis, editor vstupnej pásky, páska viditeľná počas simulácie, čítacia hlava simuluje pohyb po páske, editor prechodovej funkcie, editácia počiatočných a koncových stavov, deteminizimus, grafický výstup, simuláciu jednotlivých krokov, nastavenie rýchlosti simulácie
Turing Machine Simulator	TM	textový vstup, vstup zo súboru, editor vstupnej pásky, páska viditeľná počas simulácie, čítacia hlava simuluje pohyb po páske, editor prechodovej funkcie, editácia počiatočných a koncových stavov, deteminizimus, grafický výstup, simuláciu jednotlivých krokov, nastavenie rýchlosti simulácie
FSA Applet	FA	vstup zo súboru, grafický vstup, editor vstupnej pásky, páska viditeľná počas simulácie, čítacia hlava simuluje pohyb po páske, editácia počiatočných a koncových stavov, deteminizimus, grafický výstup, simuláciu jednotlivých krokov
Simulátor Turingova Stroje (Stehlík)	TM	textový vstup, vstup zo súboru, na obrazovke znázornený formálny zápis, editor vstupnej pásky, páska viditeľná počas simulácie, čítacia hlava simuluje pohyb po páske, možnosť meniť vstupnú pásku počas behu programu, editácia počiatočných a koncových stavov, deteminizimus, nedeteminizimus, simulácia jednotlivých krokov, podsvietenie syntaxe, vyznačenie chýb syntaxe
Pašmik	FA, PDA	grafický vstup, textový vstup, vstup zo súboru, na obrazovke znázornený formálny zápis, editor vstupnej pásky, čítacia hlava simuluje pohyb po páske, editor prechodovej funkcie, editácia počiatočných a koncových stavov, deteminizimus, nedeteminizimus, grafický výstup, simulácia jednotlivých krokov, nastavenie rýchlosti simulácie, zobrazuje strom všetkých možností, možnosť voliť nasledujúci krok pri nedeterminizme
Kohaut	FA, PDA, TM	vstup zo súboru, editor vstupnej pásky, páska viditeľná počas simulácie, čítacia hlava simuluje pohyb po páske, deteminizimus, grafický výstup, simuláciu jednotlivých krokov, vyznačenie chýb syntaxe
Porubčan	FA, PDA, TM	textový vstup, vstup zo súboru, na obrazovke znázornený formálny zápis, editor vstupnej pásky, páska viditeľná počas simulácie, čítacia hlava simuluje pohyb po páske, deteminizimus, nedeteminizimus, simuláciu jednotlivých krokov
Kuliha	FA, PDA	grafický vstup, textový vstup, vstup zo súboru, na obrazovke znázornený formálny zápis, editor vstupnej pásky, editor prechodovej funkcie, editácia počiatočných a koncových stavov, nedeteminizimus, grafický výstup, simuláciu jednotlivých krokov

Rodina	TM, RAM, AM	grafický vstup, textový vstup, vstup zo súboru, na obrazovke znázornený formálny zápis, editor vstupnej pásky, páska viditeľná počas simulácie, čítacia hlava simuluje pohyb po páske, editor prechodovej funkcie, editáciu počiatočných a koncových stavov, deteminizimus, nedeteminizimus, grafický výstup, simulácia jednotlivých krokov, nastavenie rýchlosti simulácie, podsvietenie syntaxe, zobrazenie kompletného výpočtu

Tab. 4: Funkcionalita simulátorov

2. ŠPECIFIKÁCIA POŽIADAVIEK

Táto kapitola sa zaoberá špecifikáciou a analýzou požiadaviek, ktoré by mal vytváraný simulátor spĺňať. Navrhovaný systém musí predstavovať komplexný nástroj na modelovanie a simuláciu stavových automatov. Má slúžiť ako učebná pomôcka, jeho cieľom je teda uľahčenie pochopenia danej problematiky. Dôraz musí byť preto kladený na čo najväčšiu názornosť, prehľadnosť a jednoduché používanie s minimálnou potrebou učenia sa. Z pohľadu používateľa musí byť systém rozdelený na dva ucelené nástroje, editor a simulátor.

Vytváraná aplikácia má byť aplikáciou modulárnou a ľahko rozšíriteľnou. Z tohto dôvodu je potrebné rozdeliť aplikáciu na 2 samostatné celky: moduly používateľského rozhrania a na výpočtovú logiku. Moduly používateľského rozhrania predstavujú tie časti aplikácie, ktoré prídu do priameho kontaktu s používateľom. Poskytnú mu rozhranie na editáciu a následnú simuláciu automatu. Tieto moduly nesmú obsahovať žiadnu výpočtovú logiku.

Za výpočtovú logiku musí byť zodpovedný samostatný modul, ktorého funkcionalitu budú jednotlivé moduly používateľského rozhrania využívať. Tento modul musí mať na starosti všetky operácie súvisiace so simuláciou automatu. Po vytvorení dostatočne všeobecného výpočtového jadra bude možné jednotlivé moduly používateľského rozhrania jednoducho, podľa potreby obmieňať.

Kvôli prehľadnosti sme jednotlivé požiadavky rozdelili do samostatných podkapitol. Najprv sú uvedené požiadavku pre konečný automat, zásobníkový automat a Turingov stroj. Tieto sú rozdelené na nasledujúce skupiny: grafické používateľské rozhranie, požiadavky na logiku a jadro systému, všeobecné požiadavky, a na záver požiadavky na správu súborov. V samostatných podkapitolách sú ďalej uvedené požiadavky na RAM stroj a počítadlový stroj, keďže na ne kladené požiadavky sú sčasti odlišné od ostatných typov automatov.

2.1. POŽIADAVKY NA GRAFICKÉ POUŽÍVATEĽSKÉ ROZHRANIE

Požiadavky na používateľské rozhranie vychádzajú priamo zo zadania projektu. Musí sa jednať o nástroj na simuláciu a vizualizáciu rôznych výpočtových zariadení, primárne konečných a zásobníkových automatov a Turingových strojov. Grafické používateľské rozhranie teda musí obsahovať nástroje na vizualizáciu automatov, ich simuláciu a samozrejme aj editor na

vytvorenie používateľom definovaného automatu. Konkrétne požiadavky na jednotlivé nástroje sú rozobrané v nasledujúcich podkapitolách.

V grafickom rozhraní sa musí používateľ vedieť zorientovať a mal by vedieť intuitívne rozpoznať funkcionalitu jednotlivých tlačidiel. Náš návrh používateľského rozhrania preto musí pozostávať z viacerých okien, ktoré musí byť možné jednoducho presúvať, zatvárať, skryť, meniť ich veľkosť a pod.

Používateľovi musí byť umožnené zvoliť si, čo potrebuje mať zvýraznené, väčšie, nesmie byť odkázaný len na prednastavené veľkosti, ktoré by mu nemuseli vyhovovať.

Kvôli jednoduchosti ovládania je nutné sprístupniť príslušné okná. Hlavnými oknami, ktoré musia byť viditeľné pri všetkých typoch automatov, sú zobrazenie pásky s hlavami, stavový diagram, prechodové funkcie a voľba rýchlosti simulácie.

Zobrazenie ďalších okien závisí od konkrétneho automatu (napríklad pri zásobníkovom sa zobrazí okno so zásobníkom).

2.1.1. ŠPECIFIKÁCIA EDITORA

Základnou požiadavkou na editor je prehľadnosť a jednoduchosť používania. Editor by mal byť zrozumiteľný, intuitívny, s minimálnou potrebou učiť sa ho používať. Za týmto účelom by mala byť využitá analógia s bežnými grafickými editormi. Editor musí poskytovať všetku funkcionalitu potrebnú na modelovanie daného automatu. Nemal by však obsahovať žiadnu nadbytočnú funkcionalitu, ktorá by ho robila neprehľadným. Vzhľad a funkcie editora by preto mali závisieť od konkrétneho typu automatu, ktorý je práve vytváraný.

Editor pások s hlavami

Editor pásky musí umožňovať plne editovať vstupné pásky a čítacie hlavy. Musí byť možné pridávať ľubovoľný počet pások. Obsah pások musí byť jednoducho a pohodlne editovateľný. Mal by byť znázornený rozdiel medzi konečnou a nekonečnou páskou.

Na pásku musí byť možné pridávať ľubovoľný počet hláv. Konkrétna hlava sa môže pohybovať po jednej alebo viacerých páskach. Táto skutočnosť musí byť v editore zrozumiteľne zobrazená a editovateľná. Pre každú hlavu musí byť možné určiť:

- *smer pohybu* – či sa môže hlava pohybovať len jedným smerom, alebo obojsmerne,

- *čítanie a zápis* – či je hlava schopná pásku iba čítať, alebo vie aj zapisovať.

Editor stavového diagramu

Editor musí umožňovať vytvorenie stavového diagramu v grafickom prostredí. Pre jednoduché používanie a minimálnu potrebu učenia by mala byť využitá analógia s existujúcimi grafickými editormi – kresliace programy, rôzne CASE nástroje na tvorbu diagramov, a pod. Editor musí umožňovať:

- pridávať stavy, určiť či je stav počiatočný alebo koncový,
- pridávať prechodové funkcie – medzi dvoma stavmi alebo slučku zo stavu do toho istého stavu, definovať symboly na ktoré sa daná funkcia vzťahuje,
- pridávať všetky objekty špecifické pre konkrétne typy automatov – napríklad objekty reprezentujúce posun pásky pri Turingovom stroji,
- pridané objekty dodatočne upravovať, presúvať, mazať.

Editor formálnej špecifikácie

Editor by mal zobrazovať úplnú formálnu špecifikáciu vytváraného automatu, teda

- formálny zápis automatu,
- množinu stavov,
- pracovnú abecedu,
- zásobníkovú abecedu (v prípade zásobníkového automatu),
- množinu koncových stavov.

Priamo v editore by malo byť možné upravovať prechodové funkcie. Editor by mal byť prepojený s editorom stavového diagramu – vykonaná zmena v editore formálnej špecifikácie by sa mala okamžite prejaviť na stavovom diagrame a naopak.

Editor zásobníka

Pri vytváraní zásobníkového automatu musí byť k dispozícii editor obsahu zásobníka. Tento by mal umožniť určiť obsah zásobníka pred začiatkom simulácie a definovať zásobníkovú abecedu.

Zadávanie špeciálnych symbolov

Pri modelovaní stavových automatov sa často používajú špeciálne symboly, ktoré sa nenachádzajú v štandardných znakových sadách. Napríklad pri Turingových strojoch sa využívajú rôzne podčiarknuté alebo nadčiarknuté písmená, symboly ukladané do zásobníka pri zásobníkovom automate sú často zapísané ako Z s indexom znaku ktorý reprezentujú (Z_a, Z_b).Editor musí umožniť zadávať aj takéto symboly.

2.1.2. TYPY AUTOMATOV

Vzhľad a funkcie editora sa musia meniť v závislosti od typu vytváraného automatu. Pri jednoduchých typoch automatov sa niektoré editory buď nezobrazia vôbec, alebo budú zobrazené s obmedzenou funkcionalitou.

V prípade konečného stavového automatu a zásobníkového automatu musí páska slúžiť len ako vstupné médium. Editor pásky nebude umožňovať meniť počet pások a hláv. Bude zobrazená jedna jednosmerne nekonečná páska a jedna hlava, ktorá bude môcť len čítať a bude sa pohybovať len doprava. Pri simulácii sa číta celý vstup, preto bude hlava umiestnená na začiatku pásky, a nebude sa dať v editore premiestňovať. Editor stavového diagramu nebude poskytovať objekty typu „L“ a „R“, určujúce smer pohybu hlavy.

Pri zásobníkovom automate musí editor pásky a stavového diagramu fungovať rovnako ako pri konečnom automate, ale zobrazí sa aj editor zásobníka.

Pri Turingovom stroji musí byť sprístupnený plnohodnotný editor pások, editor stavového diagramu bude poskytovať objekty reprezentujúce pohyb hlavy.

Pri návrhu vlastného automatu musia byť sprístupnené všetky editory, čo umožní definovať akýkoľvek automat.

Editor / Automat	Vstupná páska	Výstupná páska	Hlava	Zásobník	Prechodové funkcie	Stavový diagram
FA	Jednosmerne konečná	Jednosmerne konečná	viac, iba doprava	nie	áno	áno
PDA	Jednosmerne konečná	Jednosmerne konečná	viac, iba doprava	áno	áno	áno
TM	Obojsmerne	Obojsmerne	Viac,	nie	áno	áno

	nekonečná	nekonečná	obojsmerny pohyb, read, write			

Tab. 5: Editory v konkrétnych typoch automatov

2.1.3. SIMULÁTOR

Výsledná aplikácia musí zahŕňať niekoľko typov simulátorov pre rozličné výpočtové stroje vzhľadom na ich potreby pri konkrétnom druhu výpočtu. Tieto musia v prehľadnej forme zobrazovať prebiehajúcu simuláciu a získané výsledky.

Prvky na riadenie behu simulácie musia byť jasne oddelené od hlavného menu a prístupné len v čase spustenej simulácie. Grafické stvárnenie týchto prvkov by malo v používateľovi evokovať ich funkčnosť, aby bolo riadenie simulácie intuitívne aj pre menej skúseného používateľa.

V prehľadnom a nenáročnom menu musí byť používateľ schopný meniť nastavenie rýchlosti podľa svojich potrieb. Pri najrýchlejšom stupni je používateľovi zobrazený len výsledok výpočtu. Pri pomalších je možné vidieť jednotlivé prechody cez stavy automatu, zmeny na vstupnej a výstupnej páske. Nie je potrebný vysoký počet rôznych rýchlosti, stačí toľko, aby mal používateľ možnosť rozlíšiť jednotlivé stupne.

Simulátor musí v prehľadnej forme zobrazovať výber ciest pri nedeterministickej simulácii. Zobrazenie týchto ciest musí akceptovať rozsah vnímania používateľa. Nemal by byť v jednom okamihu zahltený prílišným množstvom dát. Je potrebné určiť hranicu objemu dát, ktoré je používateľ schopný naraz vnímať počas rôznych spôsobov zobrazení.

2.2. VŠEOBECNÉ POŽIADAVKY

Táto časť sa venuje popisu požiadaviek kladených na všeobecné nastavenia vytváraného simulátora. Definovali sme nasledujúce požiadavky:

Zobrazenie prázdneho symbolu (ε, možnosť zmeny)

Na zobrazovanie prázdneho znaku sa v praxi používa viacero možností. Niektoré simulátory používajú medzeru, iné malé e, vo všeobecnosti sa používa označenie epsilon „ε“.

Náš simulátor musí poskytovať možnosť výberu znaku, ktorý bude reprezentovať prázdny znak. Používateľ si na začiatku musí zvoliť tento znak z ponúkaných možností a systém musí ďalej používať len tento znak. Systém nesmie umožniť kombinovanie viacerých znakov v prechodových funkciách. Prednastavený znak musí byť „ε“.

Možnosť lokalizácie (Slovenský jazyk, možnosť dopĺňania iných jazykov)

Keďže nami vytváraný simulátor má byť všeobecne prístupný pre všetkých, musí existovať možnosť zvoliť si jazyk, v ktorom bude lokalizovaný celý simulátor. Prednastavený jazyk musí byť slovenský, keďže simulátor bude určený hlavne pre slovenských študentov. V neskorších prototypoch musí byť dopracovaná anglická verzia simulátoru.

2.3. VÝPOČTOVÁ LOGIKA

Všetky operácie s automatom musí mať na starosti výpočtová logika (jadro) simulátora. Na tomto jadre budú moduly používateľského rozhrania vykonávať operácie nad samotným automatom. Z dôvodu nutnosti simulácie rôznych druhov automatov je potrebné, aby poskytovaná funkcionalita bola všeobecná a neviazaná len na jeden presne špecifikovaný automat. Jadro musí z toho dôvodu poskytnúť na rozhraní funkcionalitu, ktorá umožní bližšie špecifikovať operačné parametre, pomocou ktorých ho bude možné nastaviť tak, aby sa simuloval používateľom zvolený druh automatu. Požiadavky na rozhranie v tomto prípade zahŕňajú:

- Možnosť špecifikácie počtu hláv
- Možnosť špecifikácie počtu pások
- Možnosť špecifikácie počtu hláv na pásku
- Možnosť priradiť pásku k jednotlivým hlavám
- Možnosť obmedziť funkcionality hláv

Strom prehľadávania do šírky, jeho funkcionalita

Jednou z kľúčových požiadaviek je poskytnutie možnosti výberu nasledujúceho kroku používateľovi. Práve preto je potrebné, aby systém dokázal generovať všetky možnosti. Pre tento

prípad je vhodné zvoliť strom možností a jeho prehľadávaním zaistiť tieto možnosti pre používateľa.

Deterministický/nedeterministický nástroj

Aplikácia musí zabezpečiť v prípade nedeterministického stroja to, aby sa ani jedna z možností nestratila a dala sa prezerať. To sa dá uskutočniť pomerne jednoducho pomocou nového vlákna, ktoré vznikne v momente nedeterminizmu a bude naďalej paralelne bežať s pôvodným vláknom. To platí rekurentne aj pre novovytvorené vlákna.

Riadenie vlákien

Počas simulácie nedeterministických automatov narazíme na stavy, z ktorých sa môže simulácia uberať viacerými cestami. Pokiaľ chceme používateľovi poskytnúť možnosť sledovať stav automatu na každej z týchto ciest, tak je potrebné, aby každá simulovaná cesta (vlákno) bola v rámci jadra simulátoru autonómne reprezentovaná. Je preto dôležité, aby riadenie simulácie mohlo prebiehať na úrovni jednotlivých vlákien, t.j. aby používateľské rozhranie mohlo jadru povedať, na ktorom vlákne má vykonať niektorý z príkazov riadenia simulácie (napr. krok na ďalší stav). Podpora nedeterminizmu na tejto úrovni umožní jeho efektívnu simuláciu a samotné jadro vďaka nej nebude zobrazovanie nedeterminizmu nijako obmedzovať. Požiadavky na výpočtovú logiku z pohľadu riadenia vlákien predstavujú:

- Jednoznačná identifikácia vykonávaných vlákien
- Podpora identifikácie vlákien na úrovni rozhrania jadra

Pokračovanie pri nedeterminizme

Vytváraný simulátor musí používateľovi umožniť rozhodnutie, akým spôsobom chce pokračovať pri dosiahnutí nedeterministického kroku. Musia byť poskytnuté nasledujúce možnosti:

- zobrazenie jednej správnej cesty,
- zobrazenie všetkých ciest,
- rozhodnúť sa, ktorou cestou sa simulácia bude uberať.

Nastavenie rýchlosti

Pre potreby učenia je potrebná malá rýchlosť výpočtu. Avšak pre potrebu samotného výpočtu je potrebná vysoká rýchlosť. Výpočtová logika preto musí vedieť zabezpečiť pomalé prechody medzi jednotlivými stavmi, ale aj rýchlo akceptovať či neakceptovať dané slovo.

2.4. SPRÁVA SÚBOROV

Kvôli efektívnejšej správe vytvorených automatov je potrebné, aby s nimi aplikácia vedela narábať vo forme súborov XML. Program musí používateľovi umožniť uložiť vytvorený automat do súboru v jednoduchej a prehľadnej forme. Rovnako musí umožniť načítať automat uložený v súbore do aktuálnej pamäti programu.

Ďalšou funkcionalitou súvisiacou so správou súborov je možnosť importovať/exportovať súbory do formy čitateľnej programom JFLAP. Jedná sa o súbory typu JFF so špecifickou štruktúrou, ktoré musí vedieť navrhovaný program otvoriť a ďalej s nimi pracovať. Rovnako musí umožňovať vytvorené automaty exportovať do súboru typu JFF.

2.5. POŽIADAVKY - POČÍTADLOVÝ STROJ

Tento automat sa podstatne líši od predchádzajúcich troch automatov (Konečný automat, Zásobníkový automat a Turingov stroj) nielen špecifikáciou ale aj odlišnou funkcionalitou. Pre spoločné črty (napr. grafické používateľské rozhranie) platia základné požiadavky špecifikované v predchádzajúcom texte. Definovali sme niekoľko hlavných požiadaviek, ktoré budú v ďalších etapách práce na projekte podrobnejšie rozpracované:

- modifikovnie programu za behu
- možnosť kedykoľvek meniť hodnoty registrov
- počet registrov je “neobmedzený”
- upozornenie na nekorektné príkazy
- vkladanie hotových príkladov zo súboru

Požiadavky na editor

Editor musí umožňovať nasledujúcu funkcionalitu:

- zobrazenie registrov
- zobrazenie editora programových príkazov
- zobrazenie simulácie programu
- jasné grafické vyjadrenie aké činnosti prebiehajú v automate

2.6. POŽIADAVKY NA RAM SIMULÁTOR

Rovnako ako v prípade počítadlového stroja, aj tento simulátor sa značne líši od prvých troch analyzovaných automatov. Pre nasledujúce požiadavky teda platia rovnaké podmienky, ako pre počítadlový stroj:

- Podporovať celú inštrukčnú sadu stroja RAM
- Podporovať všetky typy operandov inštrukcií stroja RAM
- Pamäte musia mať „nekonečnú“ veľkosť
- Aritmeticko-logická a riadiaca jednotka riadi celý výpočet
- Umožniť modifikovať program za jeho behu
- Poskytnúť hotové programy zo súboru (formáty XML, JFF)
- Povoliť ukladanie nových programov do súboru (formát XML)
- Voľba rýchlosti simulácie, aj krokovanie
- Umožniť výber inštrukcií pri písaní programu, a pri vstupnej páske zobrazuje formát vstupu (ak sa do jednej oblasti vkladá celý vstup)
- Upozornenie na nekorektné príkazy

Požiadavky na editor

Na editor pri simulátore RAM budú kladené nasledovné požiadavky:

- Musí obsahovať vstupnú a výstupnú pásku (na každej páske sa nachádza 1 hlava)
- Musí obsahovať pamäť údajov (registre) a pamäť programu
- Graficky zobraziť priebeh simulácie – aby bolo vidieť čo sa práve robí

3. NÁVRH RIEŠENIA

V tejto kapitole predstavujeme návrh riešenia vypracovaný na základe špecifikácie požiadaviek. Jednotlivé prvky návrhu sú rozdelené do samostatných podkapitol, vybrané časti obsahujú aj predbežné návrhy obrazoviek. Na prehľadnejšie zobrazenie návrhu zobrazenia determinizmu a nedeterminizmu sme použili niekoľko diagramov prípadov použitia.

Pri návrhu riešenia sme sa zamerali na konečný automat, zásobníkový automat a Turingov stroj, ale je možné dodefinovať si vlastný typ automatu.

Z pohľadu používateľa bude systém rozdelený na dva ucelené nástroje:

- editor
- simulátor

Editor umožní rýchle a pohodlné modelovanie automatu v plne grafickom prostredí. Automat bude možné vytvárať editovaním stavového diagramu, alebo pomocou prechodových funkcií. Jednotlivé editory budú navzájom previazané, takže formálna špecifikácia bude vždy zodpovedať stavovému diagramu, čo umožní používateľovi pochopiť súvislosti medzi týmito dvoma reprezentáciami stavového automatu.

Interaktívny simulátor umožní používateľovi simulovať vytvorené automaty, pričom vždy prehľadne zobrazí všetky aspekty výpočtu, v závislosti od typu simulovaného automatu. Simuláciu bude možné krokovať, alebo nechať bežať do konca, bude možné zobraziť priebeh simulácie, alebo jej aktuálny stav. Osobitná pozornosť je venovaná nedeterminizmu a jeho vizualizácii. Simulátor podľa želania používateľa buď nájde správne riešenie, alebo mu dá pri nedeterministickom kroku možnosť výberu. Ďalšou možnosťou je zobrazenie všetkých vetví výpočtu, pri zachovaní čo najvyššej prehľadnosti.

Samotná aplikácia bude modulárna, čo umožní jej jednoduché rozšírenie o ďalšie typy automatov, prípadne ďalšie spôsoby zápisu automatu, ako je tabuľka. Samozrejmosťou je možnosť perzistentného ukladania vytvorených automatov. Aplikácia tiež bude schopná importovať a exportovať súbory vytvorenými v iných podobných nástrojoch, ako napríklad JFlap.

Pri implementácii použijeme techniku agilné programovanie, ktoré nám umožní rozširovať vytvorený funkčný prototyp o ďalšiu funkcionalitu. Zároveň umožní efektívnejšie prispôsobenie sa prípadným zmenám počas vývoja. Simulátor bude implementovaný ako stand-alone aplikácia.

3.1. JADRO SYSTÉMU

Podľa špecifikovaných požiadaviek na jadro systému je potrebné navrhnúť parametre, ktoré umožňovali jadro zadefinovať pre požadovaný typ simulovaného automatu. Medzi základné atribúty, ktoré musí jadro obsahovať, patrí:

- *Maximálny počet hláv* – V prípade potreby je vhodné, aby automat umožňoval zadefinovať maximálny počet hláv, ktoré je možné definovať. Aj keď žiadny z požadovaných automatov podobné obmedzenie nemá, ide o pridanú hodnotu, ktorej implementácia je jednoduchá. Používateľ si bude môcť zvoliť automat napríklad len s jednou hlavou.
- *Maximálny počet pások* – V prípade potreby je vhodné, aby automat umožňoval zadefinovať maximálny počet pások, ktoré je možné definovať. Aj keď žiadny z požadovaných automatov podobné obmedzenie nemá, ide o pridanú hodnotu, ktorej implementácia je jednoduchá.
- *Maximálny počet hláv na jednu pásku* – V prípade potreby je možné zadefinovať maximálny počet hláv, ktoré môžu čítať jednu pásku.
- *Maximálny počet pások na jednu hlavu* – V prípade potreby je možné zadefinovať maximálny počet pások, ktoré môže čítať jedna hlava.
- *Prítomnosť zásobníka* – Možnosť povoliť, resp. zakázať prítomnosť zásobníka v danom type automatu.
- *Možnosť zadefinovať povolené typy hláv* – Špecifikácia povolených vlastností hláv predstavuje rozsiahlejší problém. Z dôvodu väčšej flexibility a všeobecnosti jadra je vhodné k nemu pristupovať samostatne. A to nasledujúcim princípom: Prvým krokom je zadefinovanie vlastností jednotlivých typov hláv. Na úrovni samotnej definície automatu sa potom definujú typy hláv, ktoré je možné do automatu umiestniť.

Hlavy v automate rovnako predstavujú entitu, ktorej funkcionalita je závislá od typu simulovaného automatu. Z tohto dôvodu je vhodné pristupovať k hlavám rovnako všeobecne ako k samotnému automatu. Obmedzenia funkcionality hláv budú špecifikované parametrami:

- *Možnosť pohybu vpred* – Zadefinuje, či je možné, aby sa hlava posúvala po páske vpred.
- *Možnosť pohybu vzad* – Zadefinuje, či je možné, aby sa hlava posúvala po páske vzad.
- *Možnosť čítania* – Zadefinuje, či je možné, aby hlava čítala pásku.
- *Možnosť zapisovania* – Zadefinuje, či je možné, aby hlava zapisovala na pásku.
- *Uzamknutie atribútov* – Ku každému atribútu bude pridelený atribút uzamknutia. V prípade, že bude daný atribút uzamknutý, tak nebude možné meniť nastavenie daného parametra používateľom. V opačnom prípade bude môcť používateľ zadefinovať daný atribút podľa potreby. Tento prístup umožní zadefinovanie všeobecných hláv, ktoré si bude môcť používateľ ľubovoľne dodefinovať bez nutnosti vytvárania vlastných typov. Rovnaký prístup zároveň umožní zamknúť parametre typov hláv a tak poskytnúť presnú funkcionalitu závislú od typu simulovaného automatu.

Pomocou definície popísaných parametrov bude možné vytvoriť šablóny jednotlivých typov automatov. Preddefinované budú 3 základné typy:

- stavový automat,
- zásobníkový automat,
- Turingov stroj.

Samotné jadro simulátora musí poskytovať rozhranie, ktoré umožní modulom používateľského rozhrania vytvoriť simulátor podľa zvolenej šablóny. Medzi základnú funkcionalitu, ktorú musí toto rozhranie poskytovať, patrí:

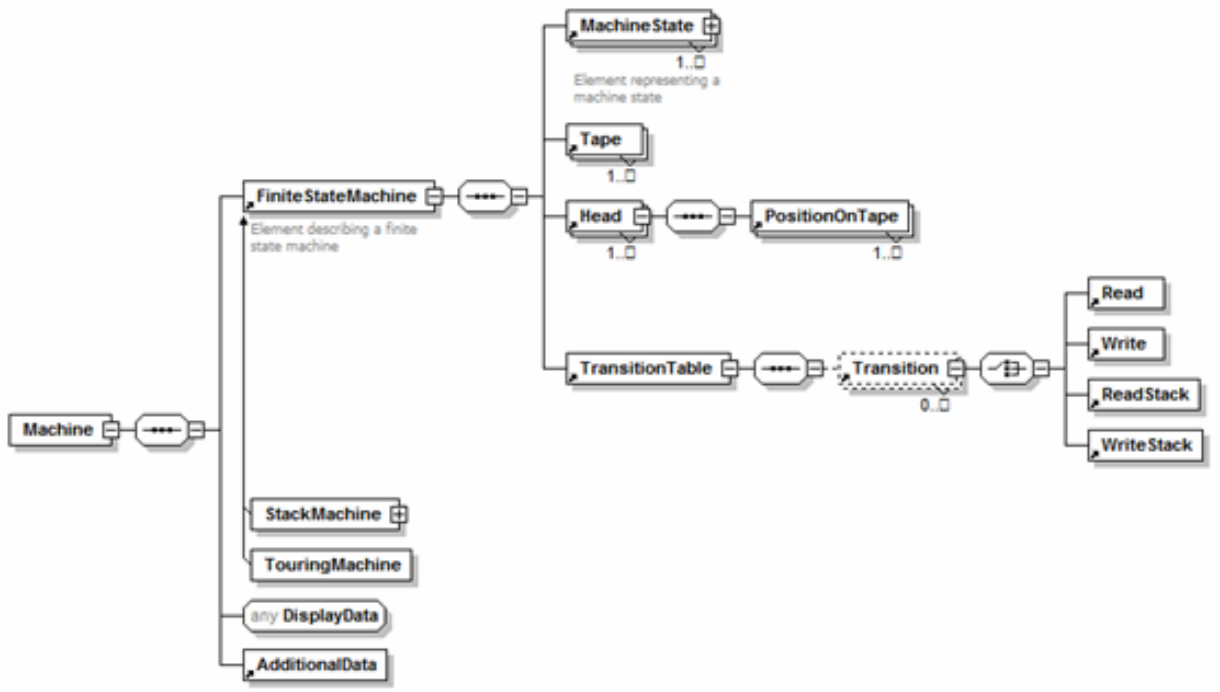
- *Definícia stavov* – Definícia jednotlivých stavov automatov.
- *Definícia prechodových funkcií* – Umožní používateľovi zadefinovať pravidlá prechodu medzi jednotlivými stavmi ako aj operácie, ktoré sa budú pri prechode vykonávať.
- *Definícia zásobníka* – V prípade, že šablóna automatu povoľuje použitie zásobníka, bude používateľ môcť zadefinovať zásobník.

- *Definícia hláv* – Umožní používateľovi vložiť do automatu hlavu podľa šablóny, prípadne editovať odomknuté parametre hlavy.
- *Definícia pások* – Umožní používateľovi vložiť pásku, prípadne meniť jej obsah.
- *Priradenie pások k jednotlivým hlavám* – Umožní používateľovi priradiť hlavy k jednotlivým páskam.
- *Riadenie simulácie* – Jadro musí poskytovať funkcionalitu, ktorá umožní modulom používateľského rozhrania riadiť priebeh simulácie. Toto riadenie zahŕňa hlavne úlohy ako spustenie simulácie, zvolenie rýchlosti simulácie, krokovanie simulácie a podobne.
- *Udalosti simulácie* – Počas simulácie sa vyskytnú udalosti, o ktorých by mali byť informované všetky moduly, ktoré danú simuláciu sledujú. Ak by bol stav simulácie odovzdávaný len prostredníctvom návratovej hodnoty metód, tak by o stave simulácie bol informovaný len ten modul, ktorý zmenu vyvolal. A to bez ohľadu na to, či je zmena predmetom jeho záujmu, alebo nie. Rôzne moduly môžu zobrazovať rôzne dôležité informácie o simulovanom automate. Z tohto dôvodu je potrebné, aby simulátor dôležité informácie oznamoval prostredníctvom udalostí.
- *Definícia registrov* – Používateľ bude môcť zadefinovať počiatočnú hodnotu registrov.
- *Definícia programových príkazov* – Definícia jednotlivých príkazov, ktoré počítadlový stroj umožňuje vykonávať
- *Riadenie simulácie* – Umožní modulom používateľského rozhrania riadiť priebeh simulácie. Toto riadenie zahŕňa hlavne úlohy ako spustenie simulácie, zvolenie rýchlosti simulácie, krokovanie simulácie a podobne.
- *Udalosti simulácie* – Počas simulácie sa vyskytnú udalosti, o ktorých by mali byť informované všetky moduly, ktoré danú simuláciu sledujú. Rôzne moduly môžu zobrazovať rôzne dôležité informácie o simulovanom automate. Z tohto dôvodu je potrebné, aby simulátor dôležité informácie oznamoval prostredníctvom udalostí.

3.2. NÁVRH XML SCHÉMY

Efektívna práca so súbormi je jednou zo základných požiadaviek definovaných na vytváraný systém. Simulátor bude pracovať so súbormi typu XML, ktoré umožňujú efektívnu a jednoduchú

správu ukladaných údajov. Predbežný návrh XML schémy je znázornený na Obr. 19. Súbory bude samozrejme možné ukladať aj vo formáte simulátora JFLAP. Medzi formátom XML používaným simulátorom a XML formátom aplikácie JFLAP bude konverziu zabezpečovať samostatný modul aplikácie.



Obr. 19: Návrh XML schémy

3.3. SIMULÁTOR

Výsledná aplikácia bude zahŕňať niekoľko typov simulátorov pre rozličné výpočtové stroje vzhľadom na ich potreby pri konkrétnom druhu výpočtu.

Spoločným znakom simulácie pre všetky tieto typy bude základné menu na ovládanie simulovania výpočtu. Spoločné menu zahŕňa položky ovplyvňujúce chod a vykonávanie simulácie. Ide o akcie *Spustiť* simuláciu, *Krokovať* simuláciu, Pozastaviť a Zastaviť simuláciu. Pod pojmom *Pozastaviť* je myslené krátke prerušenie simulácie s možnosťou opätovného pokračovania v bode zastavenia. Akcia *Zastaviť* po prerušení ukončí proces simulácie v danom bode bez možnosti pokračovať.

K spoločným položkám bude patriť aj možnosť nastavenia rýchlosti a typu simulácie. Rôzne stupne simulácie napomáhajú porozumieť danú problematiku používateľovi. Medzi

základné nastavenia patrí samostatná simulácia. Tento typ pracuje sám a nezávisí na činnosti používateľa, slúži len na určenie akceptácie alebo neakceptácie vstupu automatom. Je vhodná aj na spracovanie viacerých vstupov, kde je postupne ku každému zadanému vstupu priradené vyhodnotenie o jeho akceptácií. Pod pojmom rýchle spustenie sa tu chápe samostatná simulácia, ktorá prebieha samostatne a nepotrebuje k svojej činnosti zásah zvonka. Je tu možné nastaviť rôzne rýchlosti. Pri najrýchlejšej je používateľovi zobrazený len výsledok výpočtu, pri pomalších môže vidieť aj jednotlivé prechody cez stavy automatu, zmeny na vstupnej a výstupnej páske a pod.

Ďalším možným typom simulácie je možnosť krokovania. Prepínanie krokov simulácie vtedy závisí od používateľa. Tento mód je vhodný na pochopenie spôsobu, akým konkrétny automat dospeje k výsledku svojho výpočtu. Používateľ v každom kroku vidí zmenu stavu automatu, zmenu, ktorá sa odohrala na vstupnej páske a symbol, ktorý sa v danom okamihu zapísal na výstupnú pásku.

Drobné odlišnosti závisia na konkrétnom druhu automatu. Podľa potreby sa pridajú ďalšie štruktúry na jasné zobrazenie priebehu simulácie. V prípade zásobníkového automatu je pre názornosť dôležité simulovanie napĺňania a vyprázdňovania zásobníka.

Počítadlový stroj

Ovládanie simulácie bude rovnaké ako v predchádzajúcich automatoch. Ide o akcie Spustiť simuláciu, Krokovať simuláciu, Pozastaviť a Zastaviť simuláciu. Pod pojmom Pozastaviť je myslené krátke prerušenie simulácie s možnosťou opätovného pokračovania v bode zastavenia. Akcia Zastaviť po prerušení ukončí proces simulácie v danom bode bez možnosti pokračovať.

Pod akciou Krokovanie priebeh simulácie závisí od používateľa. Používanie tejto akcie je vhodné na pochopenie spôsobu, akým automat dospeje k výsledku svojho výpočtu. Používateľ v každom kroku vidí príkaz, ktorý sa práve vykonáva, hodnoty všetkých registrov a vyznačenie ktoré registre sa v danej chvíli menia.

RAM

Simulátor RAM stroja bude vyzerať podobne ako simulátor počítadlového stroja, samozrejme s drobnými rozdielmi:

- Zvýraznenie riadku kódu, ktorý sa práve vykonáva.

- Zmeny v registroch sa tiež zvýraznia.
- Zobrazovanie simulácie – pomocou šípok sa zobrazí postup vykonávania programu
- Vstupná a výstupná páska – 1 hlava na každej páske
- Pamäť údajov – obsahuje číslo registra a obsah v ňom. Registre majú „nekonečnú“ veľkosť
- Pamäť programu – program sa bude písať do textového poľa, v ktorom sa mu budú ponúkať názvy inštrukcií, pretože by mohlo dôjsť k nejednoznačnosti, pretože niektoré simulátory používajú mierne odlišné názvy inštrukcií (napr. MULT = MUL, JGTZ = JGZERO).
- Preprocesor – zobrazuje inštrukciu s operandom, ktorá sa práve vykonáva

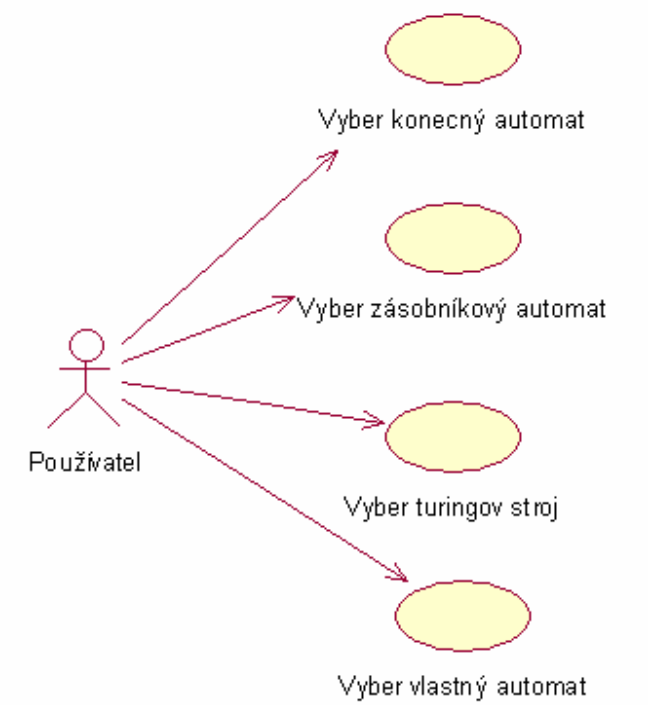
3.4. VYTVORENIE AUTOMATU

3.4.1. VYTVORENIE KONEČNÉHO A ZÁSOBNÍKOVÉHO AUTOMATU

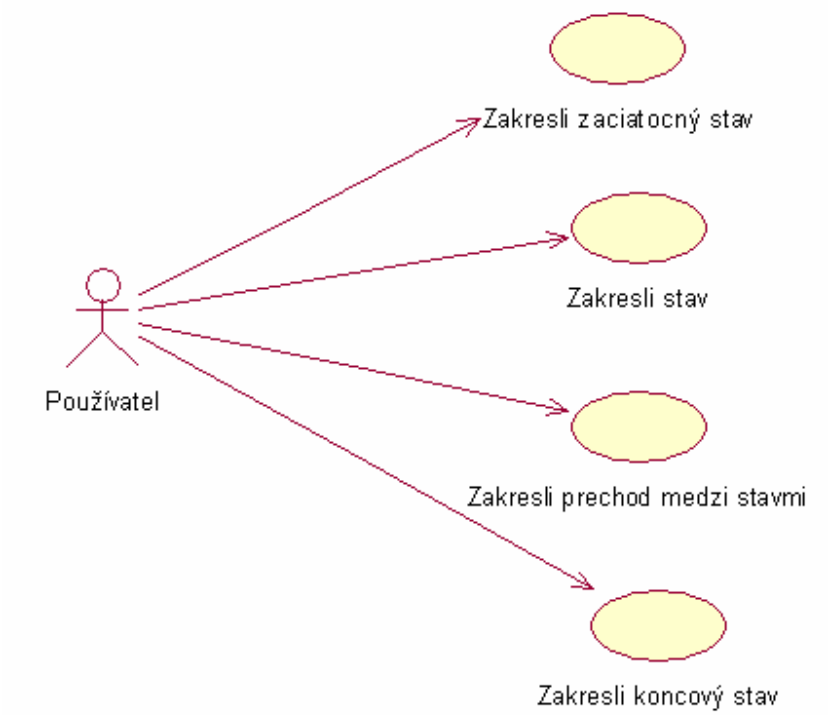
Používateľ si po spustení simulátora môže vybrať z viacerých typov automatov. Zamerali sme sa na Konečný automat, Zásobníkový automat a Turingov stroj. V ďalších etapách vývoja prototypu sa rozšíri aj o zvyšné požadované typy ako RAM a počítadlový stroj.

Podľa typu automatu si následne zvolí parametre. Tieto parametre sú prednastavené, ale používateľ si môže niektoré zvoliť tak, ako mu vyhovujú. Napríklad si môže zvoliť symbol reprezentujúci prázdny znak, počet pások, počet hláv a iné.

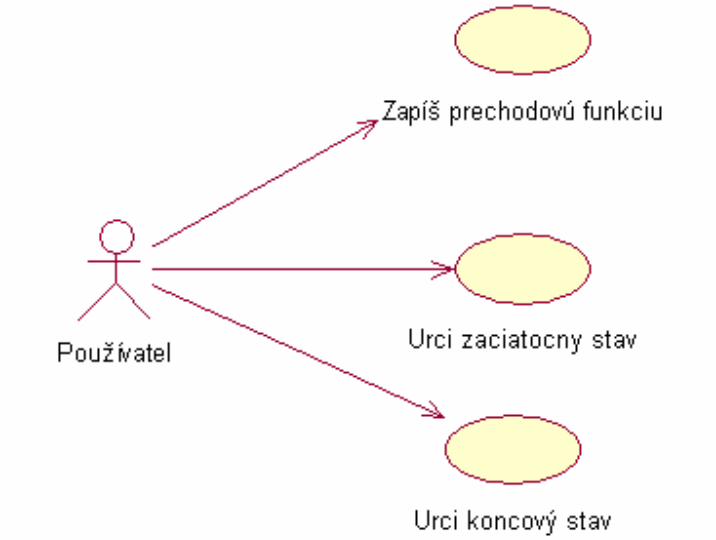
Pri navrhovaní vybraného automatu si používateľ môže zvoliť spôsob, ktorý mu najviac vyhovuje. Písanie prechodových funkcií vyžaduje znalosť zvoleného automatu a spôsob zápisu funkcií. Každý typ automatu má svoje špecifické vlastnosti, podľa ktorých sa musí riadiť. Kreslenie stavového diagramu pozostáva z kreslenia stavov a prechodmi medzi stavmi. Na záver sa musí označiť začiatočný stav a všetky koncové stavy. Diagramy prípadov použitia pre vytvorenie automatov sú znázornené na Obr. 20, Obr. 21 a Obr. 22.



Obr. 20: Diagram prípadov použitia pre výber automatu



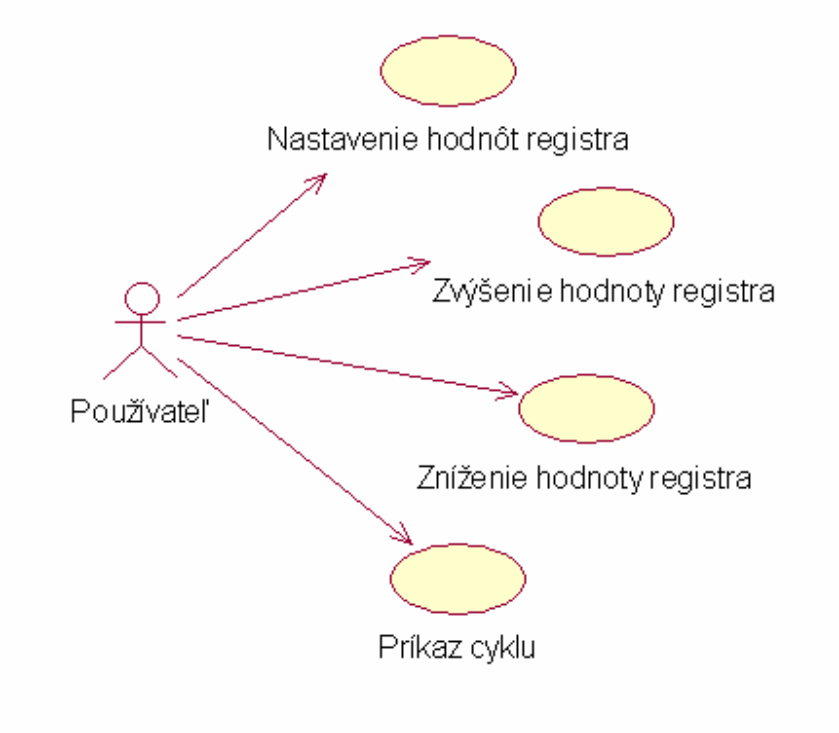
Obr. 21: Diagram prípadov použitia pre zakreslenie stavového diagramu



Obr. 22: Diagram prípadov použitia pre zapísanie prechodových funkcií

3.4.2. VYTVORENIE POČÍTADLOVÉHO STROJA

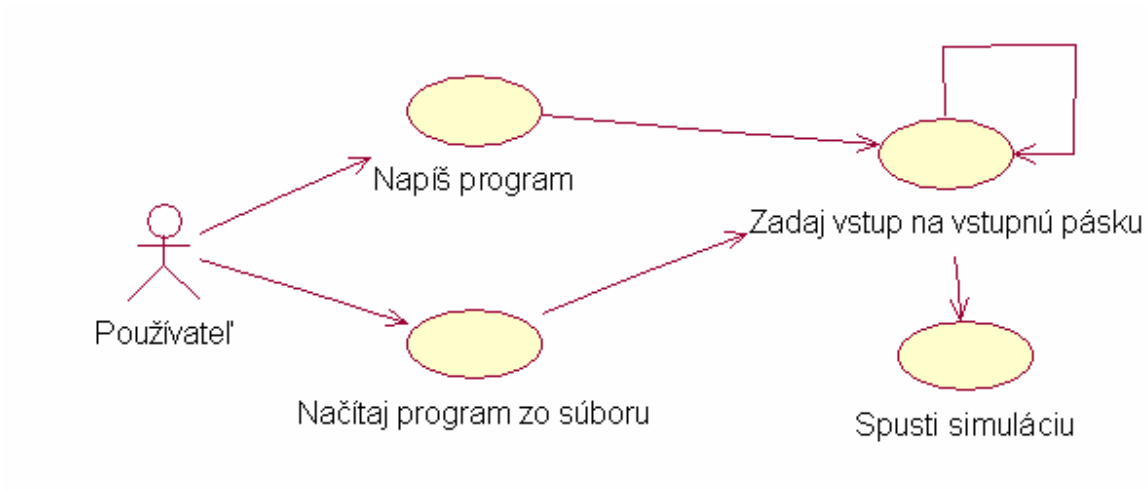
Vytváranie automatu pozostáva z nastavenia vstupných hodnôt do registrov. Vykonanie výpočtu vychádza zo série povolených príkazov, ktoré používateľ napíše. Jednoduché bloky príkazov ako presúvanie hodnôt medzi dvoma registrami, vynulovanie registra, zvýšenie alebo zníženie hodnoty registra budú vložené dopredu ako preddefinované časti kódu.



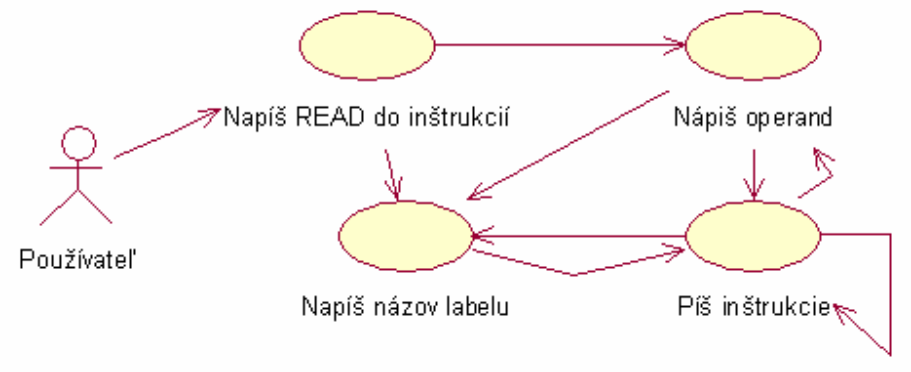
Obr. 23: Vytvorenie programu počítadlového stroja

3.4.1. VYTVORENIE RAM STROJA

Práca s RAM strojom je zobrazená na ., postup písania programu na Obr. 24.



Obr. 24: Práca RAM stroja



Obr. 25: Vytvorenie programu RAM stroja

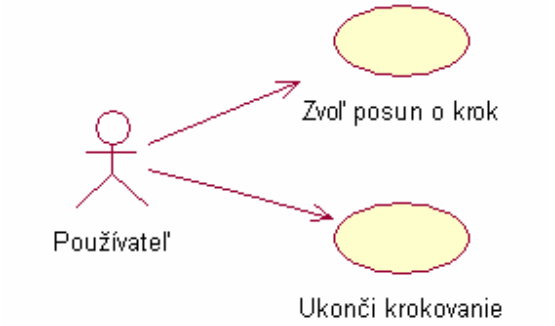
3.5. DETERMINIZMUS V SIMULÁCII

Na znázornenie deterministického automatu má používateľ možnosť vybrať si z dvoch možností.

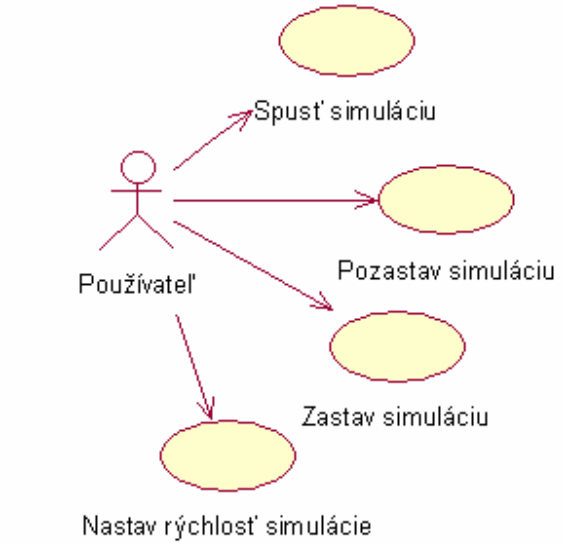
Prvou je *krokovanie simulácie*, kedy používateľ pokračuje v simulácii po jednom kroku. Tento postup poskytuje najlepší prehľad o tom, čo sa v automate vykonáva. V priebehu krokovania sa môže používateľ rozhodnúť aj pre dokončenie simulácie bez zastavenia.

Druhou možnosťou je *spustenie simulácie*, pričom na začiatku je vhodné zvoliť si rýchlosť, ktorá používateľovi najviac vyhovuje. V priebehu simulovania môže byť simulovanie pozastavené. Znamená to, že simulácia sa zastavila, pričom je možné pokračovať ďalej dokončením simulácie alebo len jej časti, prípadne prejsť na krokovú simuláciu.

Po skončení simulácie môže používateľ prejsť do zobrazenia záznamu, kde je celý priebeh simulácie vizualizovaný. Diagram prípadov použitia pre simulovanie deterministického automatu je znázornený na Obr. 26.



Obr. 26: Diagram prípadov použitia pre krokovanie simulácie



Obr. 27: Diagram prípadov použitia pre riadenie simulácie

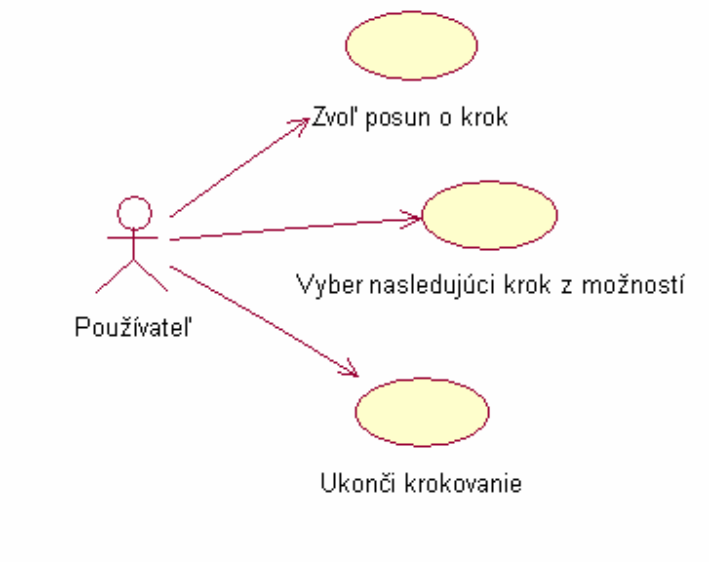
3.6. NEDETERMINIZMUS V SIMULÁCII

Na znázornenie nedeterministického automatu má používateľ možnosť vybrať si z troch možností:

1. Prvou možnosťou je *krokovanie simulácie*, kde používateľ pokračuje v simulácii po jednom kroku. Rozdiel oproti deterministickému automatu je v tom, že môže nastať situácia, kedy sa bude musieť používateľ rozhodnúť pre jednu z možností, ako pokračovať.
2. Druhou možnosťou je *spustenie automatickej simulácie*, pričom systém postupne prejde všetky vygenerované možnosti. Používateľ vidí všetky možné cesty.
3. Treťou možnosťou je, že si používateľ spustí simuláciu, ktorá mu *ukáže jednu cestu*, ktorá vedie k akceptácii. Simulátor začne generovať strom možností, ktorý sa nebude používateľovi zobrazovať. V momente, kedy nájde cestu, ktorá je akceptovaná, prehľadávanie ukončí a túto cestu zobrazí používateľovi.

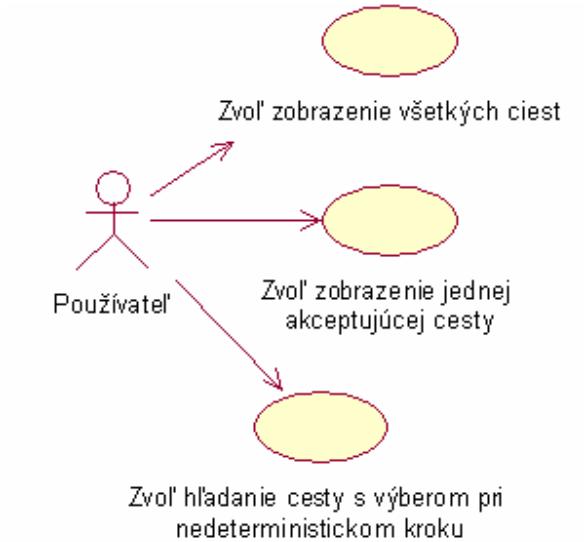
Po skončení simulácie môže používateľ prejsť do zobrazenia záznamu, kde je celý priebeh simulácie vizualizovaný. Ak je priebeh veľmi zložitý a bolo by neprehľadné zobraziť podrobnosti všetkých vlákien, budú zobrazené len najhlavnejšie body v textovom formáte. Priebeh

považujeme za veľmi zložitý vtedy, ak sa strom možností rozvetvuje do viac ako 20 vetiev. Diagram prípadov použitia pre simulovanie deterministického automatu je zobrazený na Obr. 28.



Obr. 28: Diagram prípadov použitia pre krokovací režim pri nedeterminizme

Simulácia pri nedeterminizme prebieha rovnako ako v prípade determinizmu. Diagram prípadov použitia je zobrazený na Obr. 29.



Obr. 29: Diagram prípadov použitia pre riešenie nedeterminizmu

3.7. DIAGRAM ČINNOSTI SIMULÁCIE NEDETERMINISTICKÉHO AUTOMATU

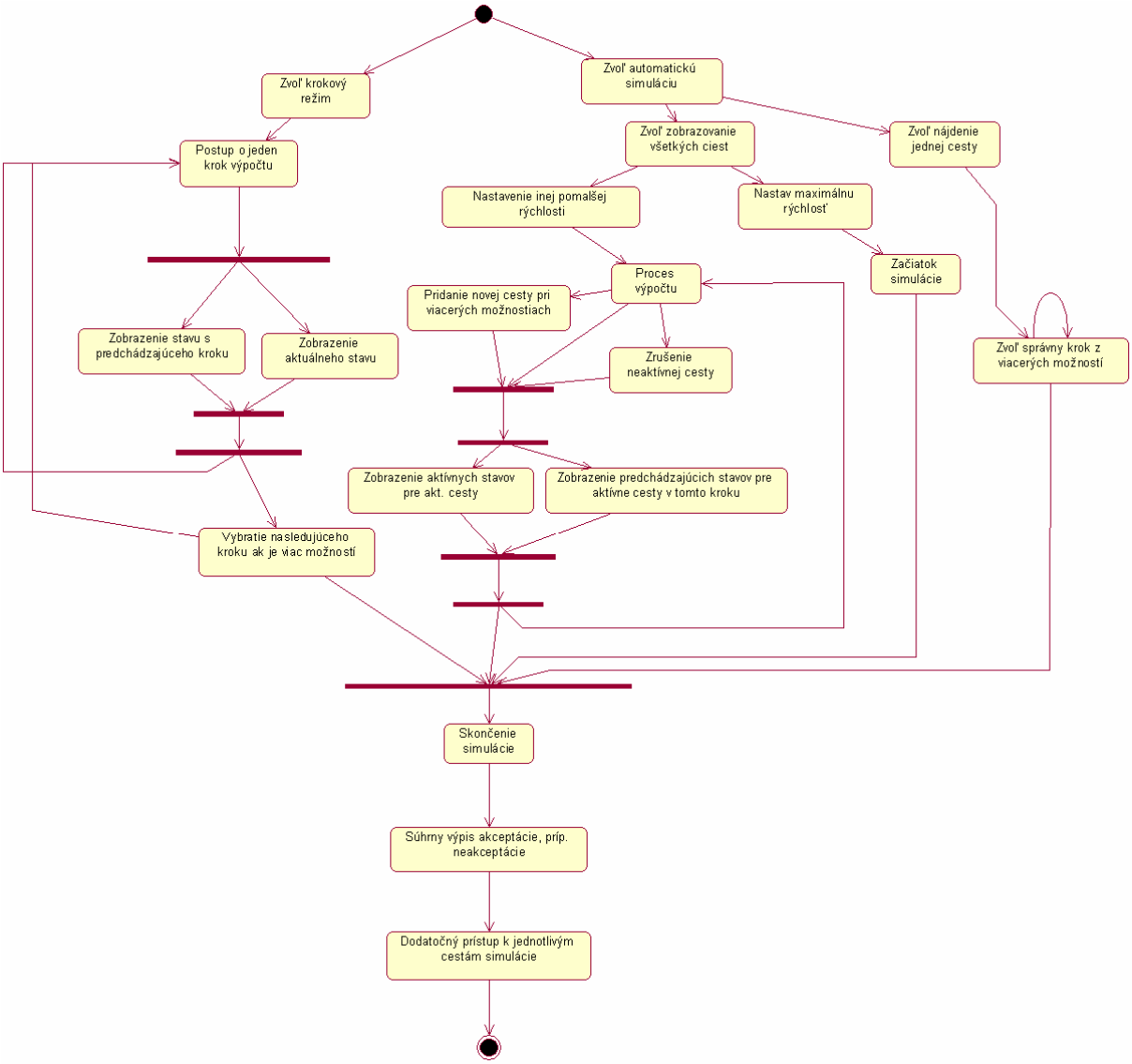
Na znázornenie nedeterminizmu automatu existuje možnosť simulácie s výberom cesty. V tomto prípade ide simulácia automaticky, prípadne po krokoch až do stavu, v ktorom dochádza k nedeterministickému rozhodovaniu. Na tomto mieste zastane a zobrazí používateľovi možnosti, ktoré sú v danom bode k dispozícii. Je na používateľovom rozhodnutí, ktorú z možností zvolí. Pri tomto spôsobe simulácie nemusí proces skončiť v akceptačnom stave, dokonca môže dôjsť aj k zacykleniu.

Ďalšou možnosťou je automatický priebeh simulácie. Po skončení sú sprístupnené všetky možné cesty, ktorými sa dalo dostať do akceptačného stavu. Túto situáciu je vhodné riešiť cez paralelné vlákna. Ak automat dôjde do stavu s nedeterministickými krokmi, vytvoria sa nové vlákna a každé pôjde svojou zvolenou cestou. Ak sa pri výpočte dostane nejaké vlákno do stavu, z ktorého nie je možné ďalej pokračovať a tento stav nie je konečný, dané vlákno sa ukončí, čím sa zároveň šetrí pamäť aj čas procesora pri spracovaní ďalšieho výpočtu. V každom kroku výpočtu sa vykoná len jeden krok každého vlákna. To znamená, že sa nespracováva najprv jedno celé vlákno a až následne ďalšie. Vďaka tomu sa simulátor nemôže pri výpočte zacykliť v jednom nekonečnom vlákne, aj keď by ostatné vlákna mohli viesť k riešeniu.

Výber správnej cesty bude riešený pomocou stromu prehľadávaného do šírky. Algoritmus na jeho prehľadávanie bude štandardný.

Poslednou možnosťou je automatická simulácia jednej cesty, krok po kroku, ale bez zásahu používateľa. Pri každom nedeterministickom kroku sa vyberie práve jedna možnosť tak, aby simulácia skončila v akceptačnom stave. Proces tejto simulácie zahŕňa jednu cestu.

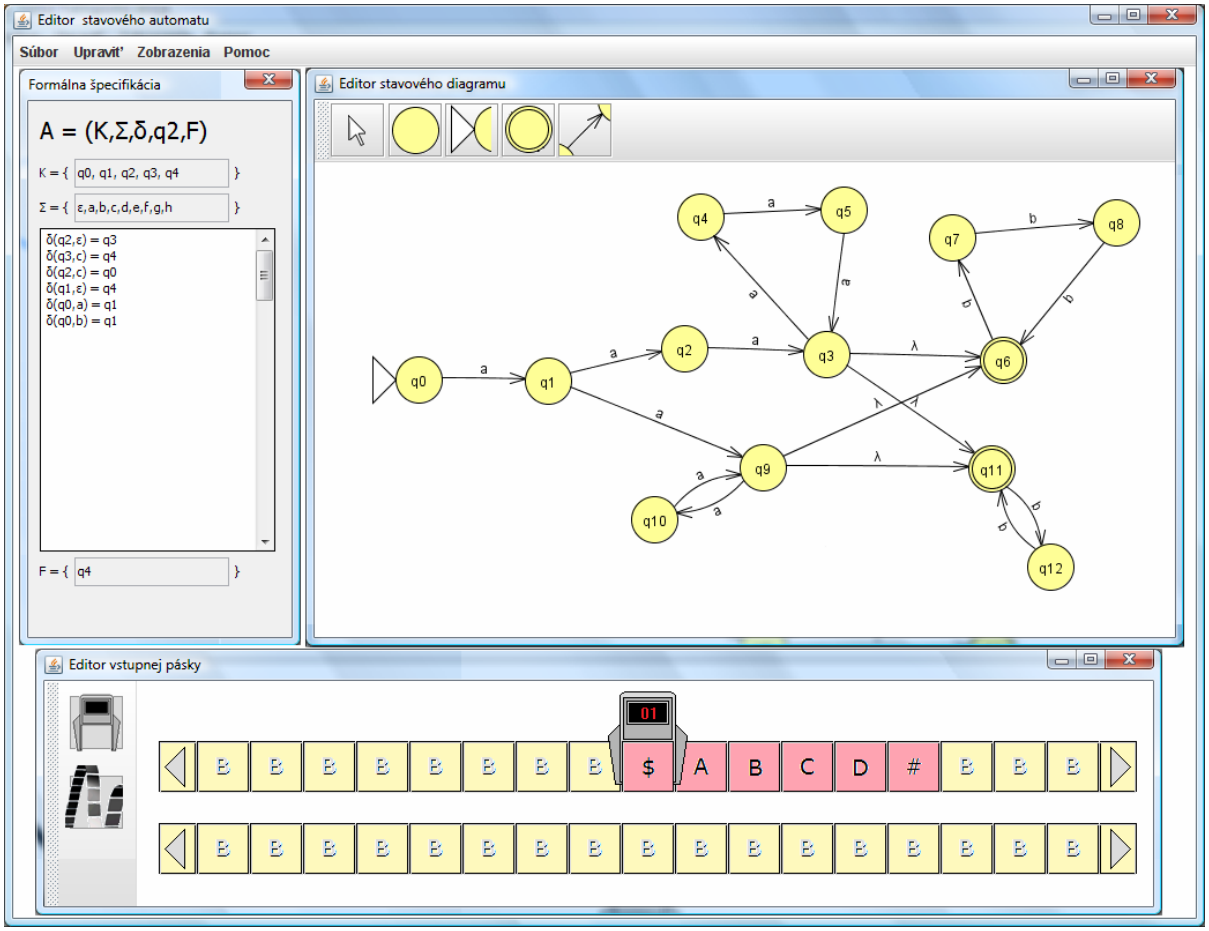
Problémom môže byť vhodné a hlavné prehľadné vykreslenie výpočtu nedeterminizmu používateľovi. Pri spôsobe, keď používateľ sám vyberá nasledujúci krok, tento problém odpadá, pretože sa zobrazuje len nasledujúci stav, ktorý si v predchádzajúcom kroku vybral. Pri automatickej simulácii, treba brať ohľad na množstvo prezentovaných rôznych vlákien ako aj názorné vykreslenie, z ktorých predchádzajúcich stavov automat prešiel do aktuálneho stavu. Diagram činnosti pri nedeterministickom automate je znázornený na Obr. 30..



Obr. 30: Diagram činnosti simulácie nedeterministického automatu

3.8. NÁVRH EDITORA

Editor bude koncipovaný ako grafická aplikácia s dôrazom na jednoduchosť používania. Bude sa skladať z niekoľkých samostatných editorov. Na zobrazenie týchto editorov bude použitý systém tzv. „ukotviteľných okien“, využitý napríklad vo vývojovom prostredí Eclipse a MS Visual Studio. Jednotlivé editory budú zobrazené ako samostatné okná v rámci hlavného okna editora. Okná sa budú dať ľubovoľne premiestňovať, škálovať, a ukotviť do rôznych častí editora. Bude poskytnuté základné rozloženie okien, pričom používateľ si môže toto rozloženie upravovať podľa svojich potrieb.



Obr. 31: Okno editora

Na Obr. 31 je znázornené hlavné okno editora s tromi editormi – editorom stavového diagramu, formálnej špecifikácie, a vstupnej pásky. Editory sa dajú ľubovoľne presúvať, dajú sa minimalizovať alebo zatvoriť.

Vzhľad a funkcie editora sa budú meniť v závislosti od typu vytváraného automatu. Pri jednoduchších automatoch sa niektoré editory buď nezobrazia vôbec, alebo budú zobrazené s obmedzenou funkcionalitou.

V prípade stavového automatu a zásobníkového automatu bude páska slúžiť len ako vstupné médium. Editor pásky nebude možňovať meniť počet pások a hláv. Bude zobrazená jedna konečná páska a jedna hlava, ktorá bude môcť len čítať a bude sa pohybovať len doprava. Pri simulácii sa číta celý vstup, preto bude hlava umiestnená na začiatku pásky, a nebude sa dať v editore premiestňovať.

Pri zásobníkovom automate bude editor pásky a stavového diagramu fungovať rovnako, zobrazí sa aj editor zásobníka.

Pri Turingovom stroji bude sprístupnený plnohodnotný editor pásky, editor stavového diagramu bude poskytovať objekty reprezentujúce pohyb hlavy.

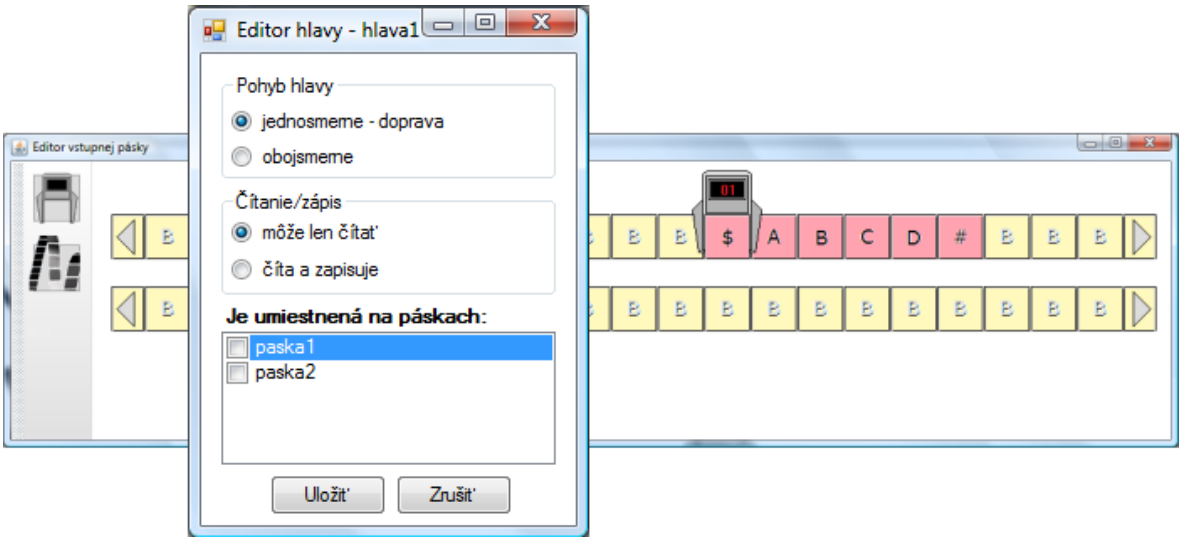
Pri návrhu vlastného automatu budú sprístupnené všetky editory, čo umožní definovať akýkoľvek automat.

3.8.1. EDITOR PÁSKY

Editor pásky bude realizovaný ako vizuálny editor, bude zobrazovať vizuálne reprezentácie editovaných objektov. Na plochu editora bude možné umiestniť ľubovoľný počet pások. Páska bude zobrazená ako reťaz políčok zobrazujúcich príslušný symbol. Tieto symboly bude možné editovať priamo kliknutím na príslušné políčko. Taktiež bude možné pridávať do pásky nové políčka a odstraňovať políčka. Tiež bude možné zobraziť si celý obsah pásky a editovať ho ako textový reťazec. Editor bude graficky a funkčne rozlišovať konečnú a nekonečnú pásku. V prípade nekonečnej pásky bude páska inicializovaná nekonečným počtom hodnôt „Blank“. Problém so správou pamäte pri nekonečnej páske bude riešený pomocou interných nástrojov jazyka Java. Pásku bude možné ľubovoľne posúvať do oboch strán.

Počet pások a hláv nebude obmedzený, musí však spĺňať podmienky, že na každej páske sa nachádza maximálne jedna hlava. Pre každú hlavu bude tiež možné určiť, či sa môže pohybovať do oboch strán, alebo len do jednej, a či môže len čítať, alebo aj zapisovať. Tieto rozdiely budú pre názornosť vyjadrené rôznymi grafickými reprezentáciami hláv. Hlava bude

zobrazovať stav v ktorom sa nachádza, prípadne stručný popis stavu. Hlavu bude možné ľubovoľne posúvať po páske.



Obr. 32: Editor pásky s pomocným editorom hlavy

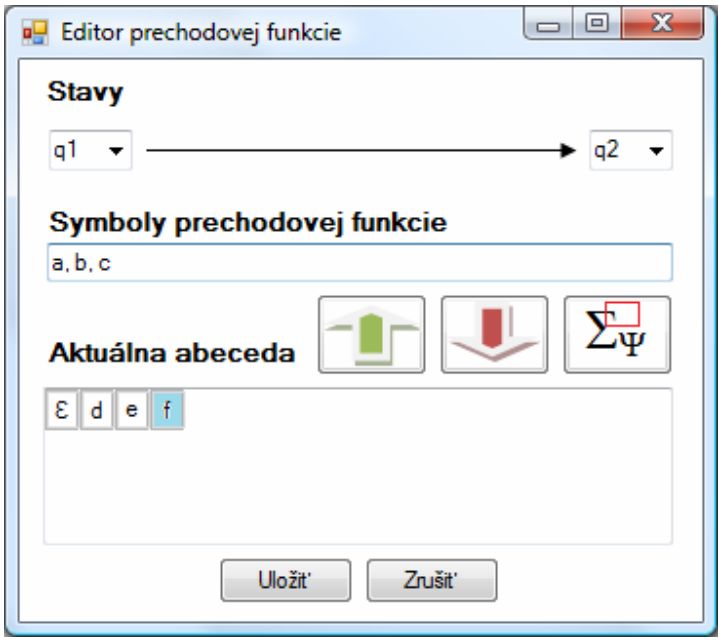
Na Obr. 32 je znázornený editor pásky s vnoreným editorom hlavy. Editor hlavy sa zobrazí pri pridávaní novej hlavy alebo pri označení existujúcej hlavy.

3.8.2. EDITOR STAVOVÉHO DIAGRAMU

Editor stavového diagramu bude pozostávať z panelu s nástrojmi a plochy s editovaným diagramom. Stavový diagram sa bude vytvárať vybraním príslušného prvku v paneli nástrojov a jeho umiestnením na plochu. Editor bude umožňovať vytvárať stavy a spájať ich prechodovými funkciami. Bude tiež umožňovať označiť počiatočný stav a koncové stavy. V prípade Turingovho stroja sa budú pridávať aj objekty reprezentujúce posun hlavy doľava alebo doprava. Po označení kurzora v paneli nástrojov bude možné objekty označovať, presúvať na ploche a zobrazovať pre ne editory. Editor stavového diagramu bude mať dva vnorené editory:

- *editor prechodovej funkcie* – zobrazí sa pri pridaní novej prechodovej funkcie, alebo pri označení existujúcej prechodovej funkcie. Bude umožňovať definovanie symbolov, pre ktoré sa daný prechod uskutoční.

- *editor stavu* – zobrazí sa pri vložení nového stavu alebo po označení existujúceho. Bude umožňovať označiť stav ako počiatočný alebo koncový. Bude tiež umožňovať pridať popis stavu.



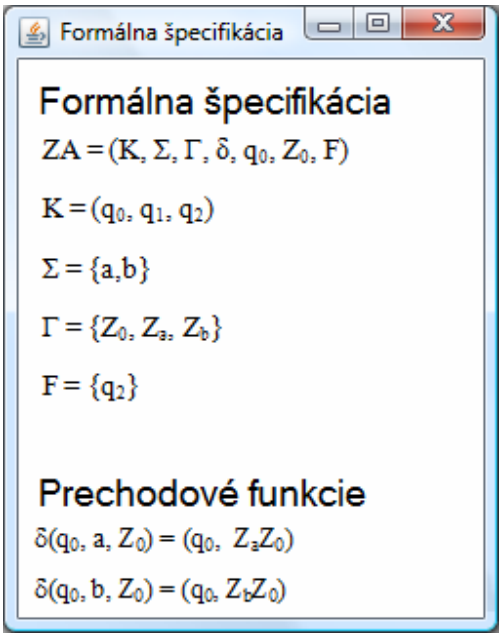
Obr. 33: Editor prechodovej funkcie

Na Obr. 33 je zobrazený editor prechodovej funkcie pre konečný automat. Vyvoláva sa z editora stavového diagramu alebo z editora formálnej špecifikácie. Umožňuje priradiť prechodovej funkcii začiatočný a cieľový stav a zadať jej symboly. Symboly je možné priamo vpísať do textového poľa alebo je ich možné pridávať zo zobrazenej abecedy. Pre prehľadnosť sú v abecede zobrazené len symboly, ktoré ešte nie sú priradené prechodovej funkcii. Z editora sa tiež otvára editor špeciálnych symbolov.

3.8.3. EDITOR FORMÁLNEJ ŠPECIFIKÁCIE

Editor bude zobrazovať formálnu špecifikáciu automatu – množinu stavov, abecedu, prechodové funkcie, počiatočný stav a množinu koncových stavov. Editor bude prepojený s editorom stavového diagramu a editorom vstupnej pásky. Zmena stavového diagramu sa okamžite zobrazí na formálnej špecifikácii. Napríklad pridanie stavu do diagramu pridá záznam o stave do množiny stavov, pridanie prechodu do diagramu pridá novú prechodovú funkciu. Zmena počtu

pások alebo hláv bude vyžadovať komplexnejšiu zmenu, musí sa zmeniť štruktúra prechodovej funkcie. Prechodové funkcie bude tiež možné editovať priamo v editore formálnej špecifikácie, zmeny sa okamžite zobrazia na stavovom diagrame.



Obr. 34: Editor formálnej špecifikácie

Obr. 34 predstavuje editor formálnej špecifikácie. Údaje v hornej časti budú generované na základe stavového diagramu. Sú zobrazené hlavne na uľahčenie pochopenia vzťahu medzi stavovým diagramom automatu a jeho formálnou špecifikáciou. Položky pod označením „Prechodové funkcie“ budú po kliknutí zobrazovať editor prechodových funkcií.

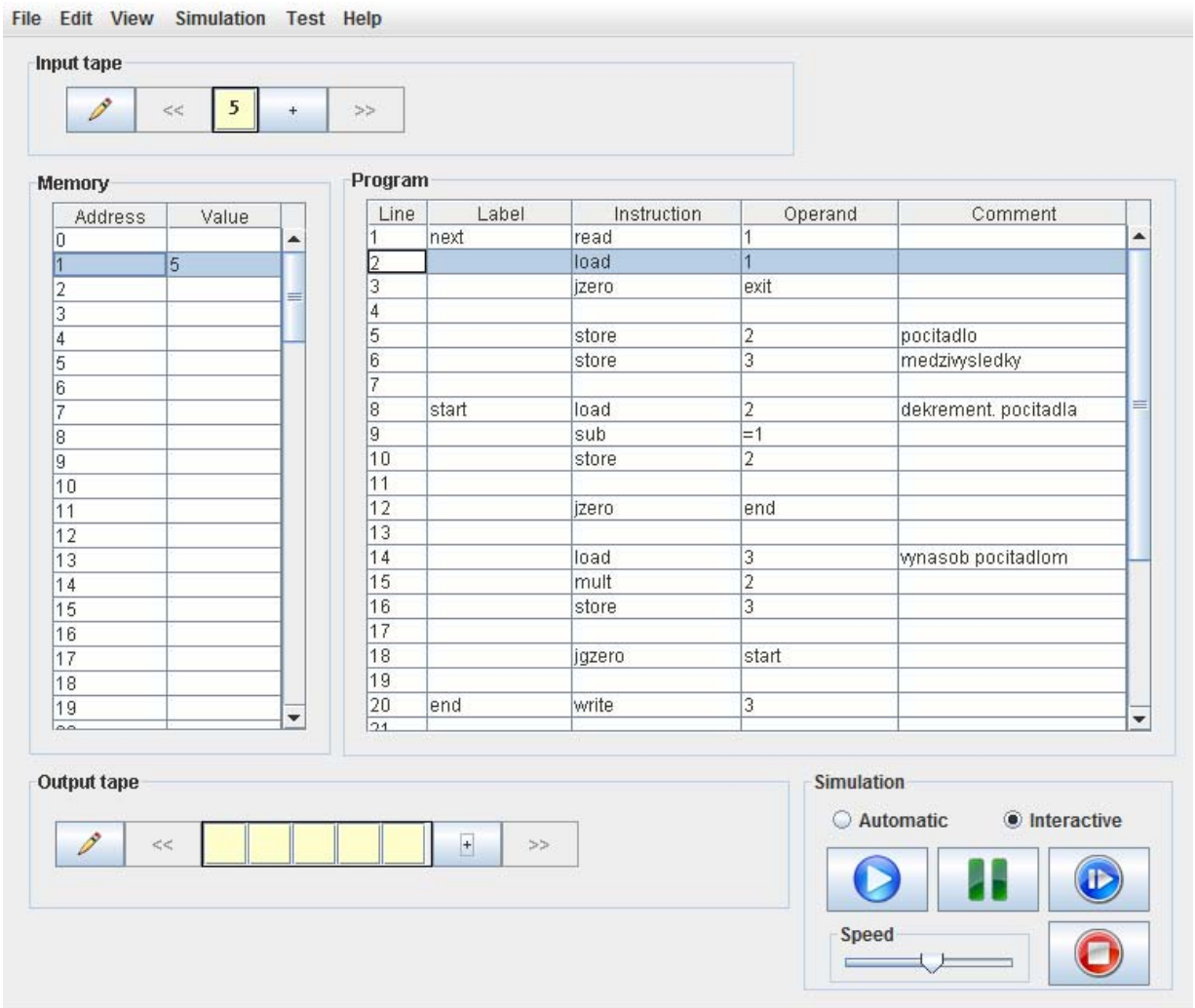
3.8.4. EDITOR ZÁSOBNÍKA

Pri vytváraní zásobníkového automatu bude k dispozícii editor zásobníka. Umožní používateľovi definovať obsah zásobníka na počiatku simulácie. Bude tiež zobrazovať aktuálnu zásobníkovú abecedu.

3.8.5. ZADÁVANIE ŠPECIÁLNYCH SYMBOLOV

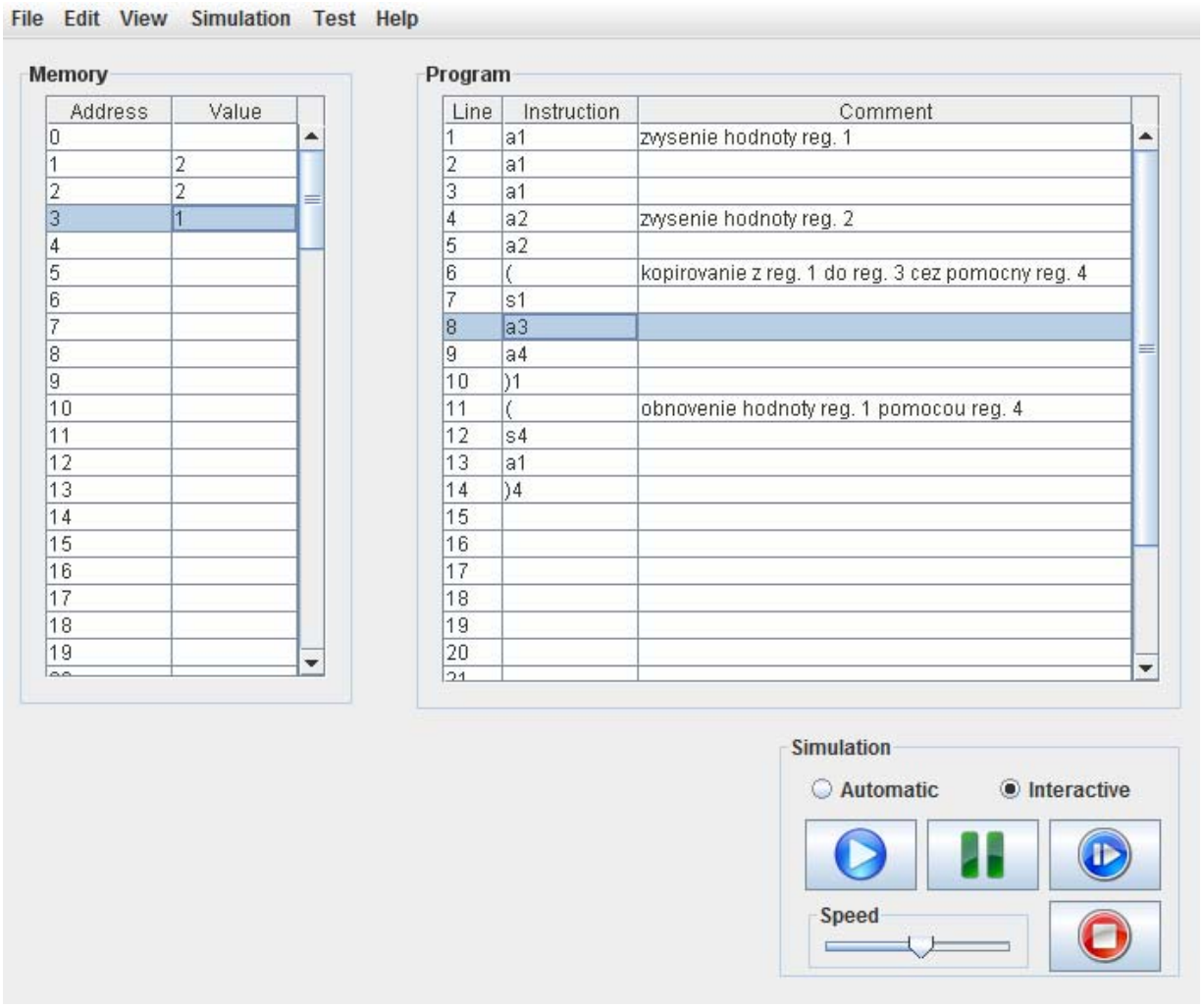
Pri modelovaní stavových automatov sa často používajú neštandardné symboly. Zadávanie takýchto symbolov bude zabezpečovať špeciálny editor. Bude prístupný z editora prechodových funkcií a z editora vstupnej pásky. Editor bude umožňovať dodefinovať si vlastné symboly – podčiarknuté, nadčiarknuté znaky, znaky s dolným a horným indexom a pod.

3.9. NÁVRH POUŽÍVATEĽSKÉHO ROZHRANIA PRE RAM STROJ



Obr. 35: Návrh GUI pre RAM stroj

3.10. NÁVRH POUŽÍVATEĽSKÉHO ROZHRANIA PRE POČÍTADLOVÝ STROJ



Obr. 36: Návrh GUI pre počítadlový stroj

4. PROTOTYP (ZIMNÝ SEMESTER)

4.1. CIEĽ PROTOTYPOVANIA

Vytváraný prototyp (ako hlavný výstup projektu v zimnom semestri) musí preukázať kvalitu navrhnutého riešenia. Jeho hlavným cieľom je dokázať, že návrh riešenia predstavuje správny postup, ktorý povedie k želaným výsledkom. Zároveň nám umožní odhaliť prípadné problémy, s ktorými sa počas implementácie môžme stretnúť. Predstavuje tak skúšku správnosti návrhu.

Keďže ako techniku vývoja aplikácie sme zvolili agilné programovanie, prototyp má pre našu nasledujúcu prácu ďalší prínos. Je ním možnosť postupne pridávať funkcionalitu a v prípade problémov nebude predstavovať problém, ak sa odkloníme od správneho riešenia.

Pri implementácii prototypu sme sa teda zamerali na aspekty, ktoré nám určia ďalšie smerovanie práce a v prípade problémov by mohli spôsobiť zmenu použitých technológií a prístupov. Konkrétne sa jednalo o vizualizáciu simulátora a celkové grafické používateľské rozhranie. Nasledujúcim krokom bolo overenie použitých knižníc pre načítavanie XML súboru, generovania tried a vizualizácie jednotlivých stavov automatu. Ďalším dôležitým bodom, od ktorého závisí ďalšie smerovanie práce, je zvolený algoritmus prehľadávania stromu. Tieto aspekty boli dôkladne otestované, výsledky sú zhrnuté v nasledujúcej kapitole.

Keďže cieľom vytvorenia prototypu nebolo implementovať všetky časti zadania, implementovali sme iba kľúčové aspekty návrhu. Ostatné časti budú dopracované v ďalších etapách práce na projekte.

4.2. DOSIAHNUTÉ VÝSLEDKY

Na overenie dôležitých aspektov prototypu sa ukázalo ako najefektívnejšie implementovať simulátor konečného automatu, ktorý je schopný načítať vstupný XML súbor, vizualizovať jednotlivé prvky simulácie a čo je najdôležitejšie, túto simuláciu vykonať. Keďže jedným z hlavných prínosov finálnej práce bude prehľadné zobrazenie nedeterminizmu, tomuto aspektu sme sa venovali aj vo vytvorenom prototype. Podarilo sa nám implementovať a overiť dva spôsoby zobrazenia nedeterminizmu. V prvom je voľba nasledujúceho kroku ponechaná na používateľovi, pri druhom spôsobe simulátor sám vyhľadá prvú cestu, ktorá povedie k správnemu výsledku a vypíše ju používateľovi. Samotné vyhľadávanie je realizované pomocou

prehľadávania stromu do šírky. Konkrétne dosiahnuté výsledky testovania sme rozdelili podľa aspektov, ktorých sa týkajú. Bližšie sú predstavené v nasledujúcich podkapitolách.

4.2.1. GRAFICKÉ POUŽÍVATEĽSKÉ ROZHRANIE

Prvá oblasť predstavuje komunikáciu aplikácie s používateľom. V tejto oblasti sme si v rámci prototypu vyskúšali, či nami špecifikované metódy prezentácie automatu, jeho jednotlivých častí, ako aj metódy zobrazenia simulácie a nedeterminizmu, sú pre používateľa dostatočne intuitívne. Práca s prototypom je intuitívna a jednoduchá. Obmedzená funkcionalita prototypu ale nedovoľuje overiť všetky zvolené princípy. Tie, ktoré bolo možné overiť sa ukázali ako správne a pre používateľa ľahko pochopiteľné. Princípy komunikácie s používateľom použité v prototype budú použité aj pri finálnej implementácií simulátora.

4.2.2. TECHNICKÁ STRÁNKA

Druhá oblasť výsledkov predstavuje technickú stránku prototypu. Za jeho pomoci sme boli schopní overiť, či je možné zostrojiť fungujúci simulátor na zvolenej platforme a za použitia zvolených technológií. Platforma Java sa ukázala ako vhodná pre implementáciu simulátora automatov.

Návrh za pomoci *XML schém* sa ukázal ako efektívny spôsob definície vlastností simulátora. S použitím XML schém úzko súvisí použitie *knižnice JAXB* v aplikácii. JAXB umožňuje generovanie Java tried z XML schém a následne poskytuje funkcionalitu na deserializáciu XML súborov do generovaných tried. Týmto spôsobom je možné definíciu automatu ukladať do textových XML súborov, ktoré sú relatívne dobre čitateľné aj za pomoci obyčajného textového editora. V prototype je použitá aj knižnica *springframework*, ktorá mimo iných funkcií umožňuje vytváranie inštancií tried s vlastnosťami nastavenými podľa preddefinovaných XML šablón. Prvotná idea použitia springframeworku bola pri vytváraní konfigurácií jadra, kde by sa inštancia triedy, ktorá definuje vlastnosti simulátora, vytvárala za pomoci springframeworku. Od tejto metódy sa počas implementácie ustúpilo a bola použitá metóda za použitia XML schémy a generovaných tried, pričom parametre simulátora sú uložené

v XML šablónach, ktoré sú deserializované za využitia JAXB. Funkčnosť tohto riešenia bola overená na schéme konečného automatu, ktorý je prototyp schopný odsimulovať.

Používateľské rozhranie bolo vytvorené za pomoci knižnice *JUNG*. Aj keď sa počas implementácie objavili problémy týkajúce sa nedostatočnej a chybnej dokumentácie tejto knižnice, jej použitie sa ukázalo ako prínos. Pomocou nej sú v prototype vizualizované jednotlivé stavy automatu.

Celkovo sa voľby platforiem a technológií ukázali ako správne a poskytujú dobrý predpoklad, že sa za ich pomoci podarí implementovať simulátor v plnom rozsahu.

4.2.3. ALGORITMUS PREHĽADÁVANIA STROMU

Ďalšiu oblasť výsledkov predstavuje funkčnosť zvolených algoritmov. Prototyp reprezentuje postupnosť stavov ako stromovú štruktúru, pričom pri hľadaní úspešného riešenie prehľadáva strom do šírky. Zvolený algoritmus sa ukázal ako efektívny a bude použitý aj pri konečnej implementácii simulátora. Simulátor je rovnako schopný identifikovať jednotlivé vetvy stromu a nezávisle ich pri simulácií rozvetvovať. Táto možnosť nie je v prototype plne využitá, ale pri finálnej implementácií sa vďaka nej budú dať používateľovi poskytnúť širšie možnosti krokovania nedeterministických automatov, a to tým spôsobom, že umožní užívateľovi dynamicky prepínať jednotlivé vetvy stromu.

4.2.4. ZHODNOTENIE DOSIAHNUTÝCH VÝSLEDKOV

Ako celok je vytvorený prototyp schopný načítať pripravený súbor obsahujúci definíciu konečného automatu. Tento automat je schopný graficky vizualizovať. Následne je používateľ schopný odsimulovať automat, a to aj spôsobom krokovania, ako aj rýchlou simuláciou automatu, kedy simulátor priamo informuje či automat vstup akceptoval. Riešene nedeterminizmu je závislé od zvolenej metódy simulácie. Pri krokovaní používateľ určí, do ktorého stavu sa automat prepne. Pri rýchlej simulácií simulátor sám zvolí tú cestu, ktorá vedie k akceptovaniu vstupu automatom.

Celkovo je implementovaný prototyp funkčný, ukazuje správnosť zvolených prístupov, technológií a algoritmov a predstavuje základný kameň pred implementáciou plnohodnotného simulátora.

5. POUŽÍVATEĽSKÁ PRÍRUČKA PROTOTYPU

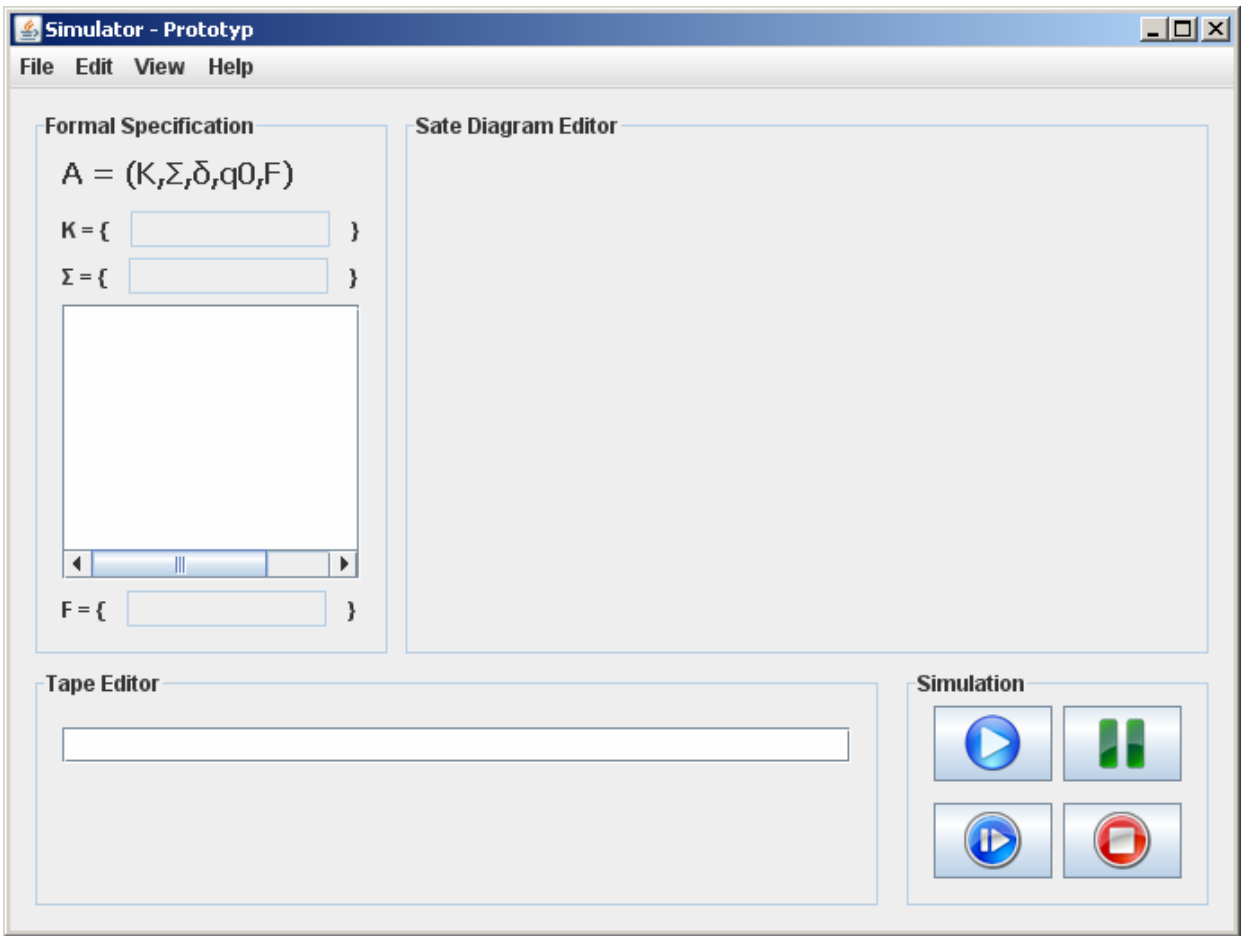
Táto kapitola opisuje prácu s prototypom simulátora. Jasne popisuje jednotlivé kroky potrebné pre prácu s týmto prototypom.

5.1. SPUSTENIE APLIKÁCIE

Otvorte príkazový riadok: Start → Run, tu napíšte `cmd`.

Do príkazového riadku napíšte `java -jar Simulator.jar` .

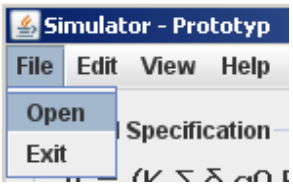
Po úspešnom spustení sa zobrazí hlavné okno, ktoré je zobrazené na Obr. 37.



Obr. 37: Hlavné okno aplikácie

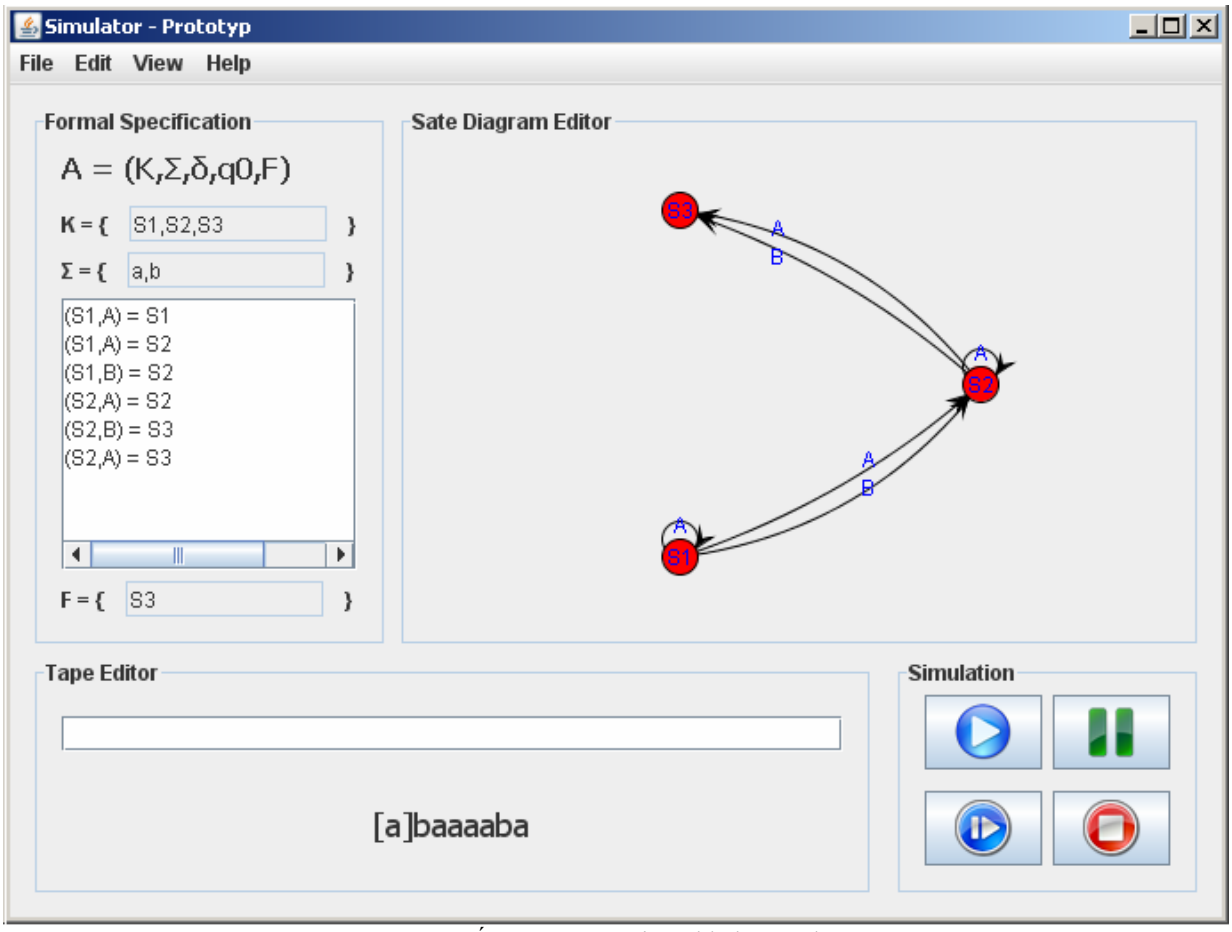
5.2. NAČÍTANIE PRÍKLADU ZO SÚBORU

Ak chcete načítať XML súbor, kliknete na File → Open.



Obr. 38: Otvorenie súboru

Následne sa zobrazí okno, kde si vyberiete XML súbor (príklad), ktorý chcete načítať. Po úspešnom načítaní sa zobrazí správa o úspešnom načítaní a v hlavnom menu sa vykreslia jednotlivé stavy, prechodové funkcie, graf a vstupná páska.



Obr. 39: Úspešné načítanie príkladu zo súboru

5.3. SPUSTENIE SIMULÁCIE

Na spustenie simulácie vyberte jednu z nasledujúcich možností:

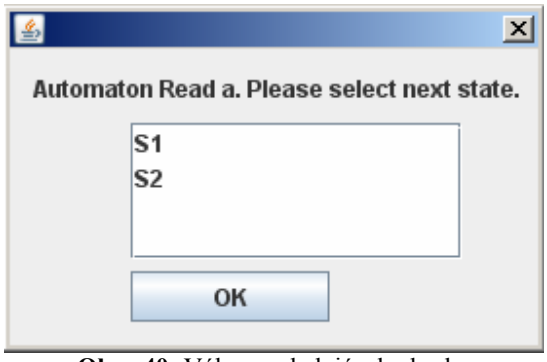
1. Ak chcete spustiť simulovanie bez zastavenia, stlačte tlačidlo „Run“. 
2. Ak máte záujem o simuláciu prostredníctvom jednotlivých krokov, stlačte pre každý nasledujúci krok tlačidlo „Step“. 

Pre pozastavenie simulácie stlačte tlačidlo „Pause“. 

Ak chcete reštartovať simuláciu, stlačte tlačidlo „Stop“. 

5.4. VÝBER NASLEDUJÚCEHO KROKU

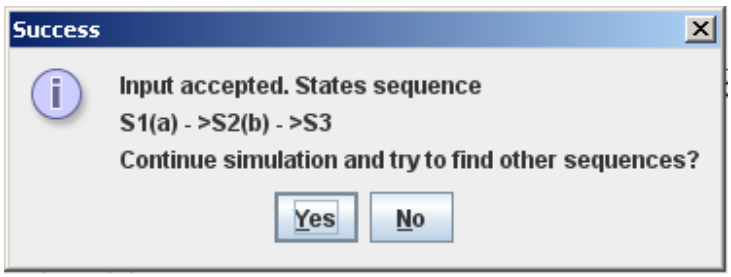
Ak existujú viaceré možnosti na prechod do ďalšieho stavu, zobrazí sa okno, v ktorom používateľ určí nasledujúci stav.



Obr. 40: Výber nasledujúceho kroku

5.5. AKCEPTOVANIE

Ak simulátor akceptuje vstup, tak prostredníctvom dialógového okna túto skutočnosť oznámi používateľovi. Taktiež mu ponúkne možnosť pokračovať v simulácii a hľadať inú sekvenciu krokov.



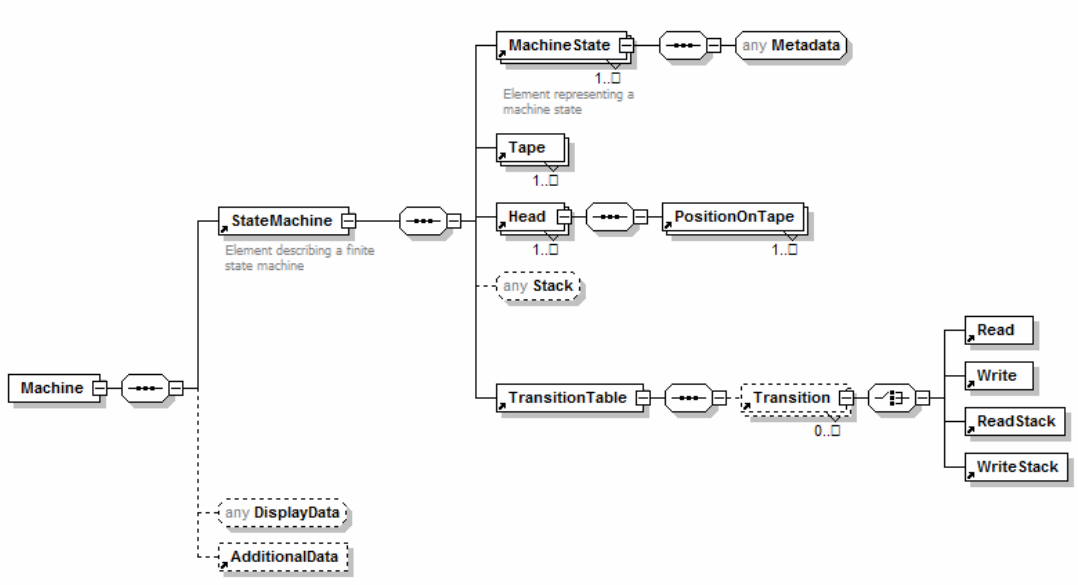
Obr. 41: Akceptovanie vstupu automatom

5.6. POHYB V GRAFE

Pre posun grafu v GUI ho jednoducho ľavým tlačidlom na myške označíte, držíte a presuniete na zvolené miesto. Ak chcete graf priblížiť alebo oddialiť, použite koliesko na myške.

5.7. POPIS XML SCHÉMY

V tejto časti sa venujeme popisu XML schémy použitej v prototype simulátora. Na Obr. 42 je zobrazená schéma popisujúca vytváraný automat.



Obr. 42: XML schéma popisujúca automat

Machine

Hlavný element, ktorý reprezentuje celý automat.

Display Data

Dáta používané pri zobrazovaní automatu. V prototype neimplementované.

Additional Data

Prídavné dáta. V prototype neimplementované.

State Machine

Reprezentácia stavového automatu. Má nasledujúce atribúty:

- *Alphabet* – Abeceda s ktorou automat operuje.
- *InitialState* – Názov počiatočného stavu automatu.
- *TemplateName* – Názov šablóny, ktorá špecifikuje operačné parametre automatu. V prototype je podporované len šablóna „Konecny Automat“.

Machine State

Reprezentácia stavu automatu. Automat môže mať nula až N stavov. Má nasledujúce atribúty:

- *isFinalState* – označuje či je stav konečný,
- *stateName* – jedinečný identifikátor, názov stavu,
- *Remark* – dodatočný popis k stavu. V prototype neimplementované.

Machine state môže mať podľa schémy element *MetaData*, tento element bude obsahovať prídavné informácie, v prototype ale nie je implementovaný.

Tape

Definícia pások automatu. Má nasledujúce atribúty:

- *Content* – prvotný obsah pásky,
- *tapeId* – jedinečný identifikátor, číslo pásky.

Head

Definícia hláv automatu. Má nasledujúce atribút:

- *TemplateName* – názov šablóny hlavy. V prototype nepodporované.

PositionOnTape

Určuje pozíciu hlavy na páske, má nasledujúce atribúty:

- *Position* – pozícia hlavy na páske. 0 určuje počiatok pásky,
- *tapeID* – identifikátor pásky, na ktorej je hlava umiestnená.

Stack

Zásobník automatu, v prototype nepodporované.

TransitionTable

Prechodová tabuľka, je zložená z elementov *Transition*, ktoré popisujú jednotlivé prechody.

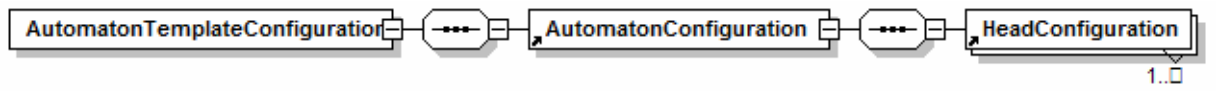
Transition

Popisuje prechod medzi dvoma stavmi. Element má nasledujúce atribúty:

- *fromState* –určuje počiatočný stav,
- *toState* – určuje cieľový stav.

V prototype je z naviazaných elementov podporovaný len element **Read**, ktorý má jeden atribút *character*, ktorý určuje znak, pri ktorého načítaní z pásky sa môže automat preklopiť z počiatočného do cieľového stavu.

Template.XSD popisuje schému pre šablóny automatov. Šablóny určujú operačné parametre automatov. Definujú ktoré operácie je automat schopný vykonať a ktoré naopak nie.



Obr. 43: XML schéma šablóny automatov

AutomatonTemplateConfiguration

Koreňový element, obsahuje jeden atribút *description*, ktorý obsahuje bližší popis šablóny pre používateľa.

AutomatonConfiguration

Špecifikuje operačné parametre automatu. Má viacero atribútov:

- *MaxHeadCount* – Definuje maximálny počet hláv, ktoré môže automat obsahovať,
- *MaxTapeCount* – Definuje maximálny počet pások, ktoré môže automat obsahovať,
- *MaxTapesForHead* – Definuje maximálny počet pások, ktoré môže čítať jedná hlava,
- *MaxHeadForTape* – Definuje maximálny počet hláv, ktoré môžu čítať jednu pásku,
- *GotStack* – Definuje, či automat obsahuje zásobník.

HeadConfiguration

Špecifikuje operačné parametre hláv automatu. Obsahuje viacero atribútov:

- *CanMoveForward* – špecifikuje, či sa hlava môže pohybovať vpred,
- *CanMoveBackwards* – špecifikuje, či sa hlava môže pohybovať vzad,
- *CanRead* – špecifikuje, či hlava môže čítať,
- *CanWrite* – špecifikuje, či hlava môže zapisovať,
- *HeadName* – Pomenovanie šablóny hlavy, ktorá je definovaná týmto elementom.

6. TESTOVANIE PROTOTYPU

Nasledujúca kapitola popisuje testovanie vytvoreného prototypu. Obsahuje testovacie prípady pre základnú funkcionalitu simulátora. Na ich základe sme overili funkčnosť navrhovaného prototypu a použijeme ich aj pri testovaní finálneho riešenia. Jednotlivé položky testovacích prípadov sú opísané a vysvetlené v tabuľke 6.

TC_ID	Testovací prípad
Popis	Názov, stručný popis testovacieho prípadu
Nastavenie	Nastavenie systému do počiatočného stavu, aby bolo možné vykonať test
Vykonané kroky	Postupnosť krokov testovacieho prípadu
Očakávaný výsledok	Popis očakávaného výsledku testu
Stav podmienky	Vyhovuje – výsledok testovania zodpovedá očakávanému výsledku

Tab. 6: Vysvetlivky k prípadom použitia

TC1 – Načítanie vstupného súboru

TC_ID	TC1
Popis	Načítanie vstupného súboru
Nastavenie	Spustenie aplikácie
Vykonané kroky	Kliknúť na File -> Open Vybrať súbor Priklad.xml, kliknúť na Open
Očakávaný výsledok	Zobrazí sa stavový diagram reprezentujúci načítaný automat Zobrazí sa formálna špecifikácia načítaného automatu Zobrazí sa vstupná páska
Stav podmienky	Vyhovuje

TC2 – Krokovanie simulácie, deterministický krok

TC_ID	TC2
Popis	Krokovanie simulácie, deterministický krok
Nastavenie	Načítanie vstupného súboru
Vykonané kroky	Stlačiť tlačidlo “krok”
Očakávaný výsledok	Stav automatu sa zmení podľa príslušnej prechodovej funkcie Čítacia hlava sa posunie na ďalší symbol
Stav podmienky	Vyhovuje

TC3 – Krokovanie simulácie, nedeterministický krok

TC_ID	TC3
Popis	Krokovanie simulácie, nedeterministický krok
Nastavenie	Načítanie vstupného súboru, krokovanie simulácie tak, aby bol nasledujúci krok nedeterministický
Vykonané kroky	Stlačiť tlačidlo “krok”
Očakávaný výsledok	Zobrazí sa dialog pre výber nového stavu. Ponúknuté stavy budú zodpovedať prechodovým funkciám pre aktuálny stav a vstupný symbol
Stav podmienky	Vyhovuje

TC4 – Výber stavu pri nedeterministickom kroku

TC_ID	TC4
Popis	Výber stavu pri nedeterministickom kroku
Nastavenie	Načítanie vstupného súboru, stlačenie tlačidla “krok” pri nedeterministickom kroku
Vykonané kroky	Vybrať jeden z ponúknutých stavov, stlačiť Ok
Očakávaný výsledok	Stav automatu sa zmení na vybraný stav Čítacia hlava sa posunie na ďalší symbol
Stav podmienky	Vyhovuje

TC5 – Spustenie simulácie do konca

TC_ID	TC5
Popis	Spustenie simulácie do konca
Nastavenie	Načítanie vstupného súboru
Vykonané kroky	Stlačiť tlačidlo “play”
Očakávaný výsledok	Simulácia prebehne do konca Ak existuje aspoň jedno správne riešenie, zobrazí sa postupnosť prechodov pre toto riešenie Používateľ má možnosť pokračovať v hľadaní ďalších riešení
Stav podmienky	Vyhovuje

TC6 – Resetovanie simulácie

TC_ID	TC6
Popis	Resetovanie simulácie
Nastavenie	Načítanie vstupného súboru, priebeh simulácie krokovaním alebo dokonca
Vykonané kroky	Stlačiť tlačidlo “stop”
Očakávaný výsledok	Stav automatu sa zmení na počiatočný stav Čítacia hlava sa presunie na začiatok vstupu
Stav podmienky	Vyhovuje

7. IMPLEMENTÁCIA PRODUKTU

Pri implementovaní výslednej podoby aplikácie sme nadviazali na prototyp vytvorený v zimnom semestri, keďže sme si ako techniku vývoja zvolili agilné programovanie. Prototyp bolo samozrejme potrebné doladiť, dopracovať zvyšnú funkcionalitu jadra a rozšíriť ho o ďalšie typy automatov. Toto sa nám úspešne podarilo; proces je opísaný v nasledujúcom texte. Simulátor sme nazvali K-I-K Simulator 2009. Vytvorená aplikácie je značne komplexná a podrobné technické opísanie je nad rámec tohto dokumentu. Sústredili sme sa preto iba na najzaujímavejšie aspekty implementácie.

7.1. TECHNICKÁ STRÁNKA

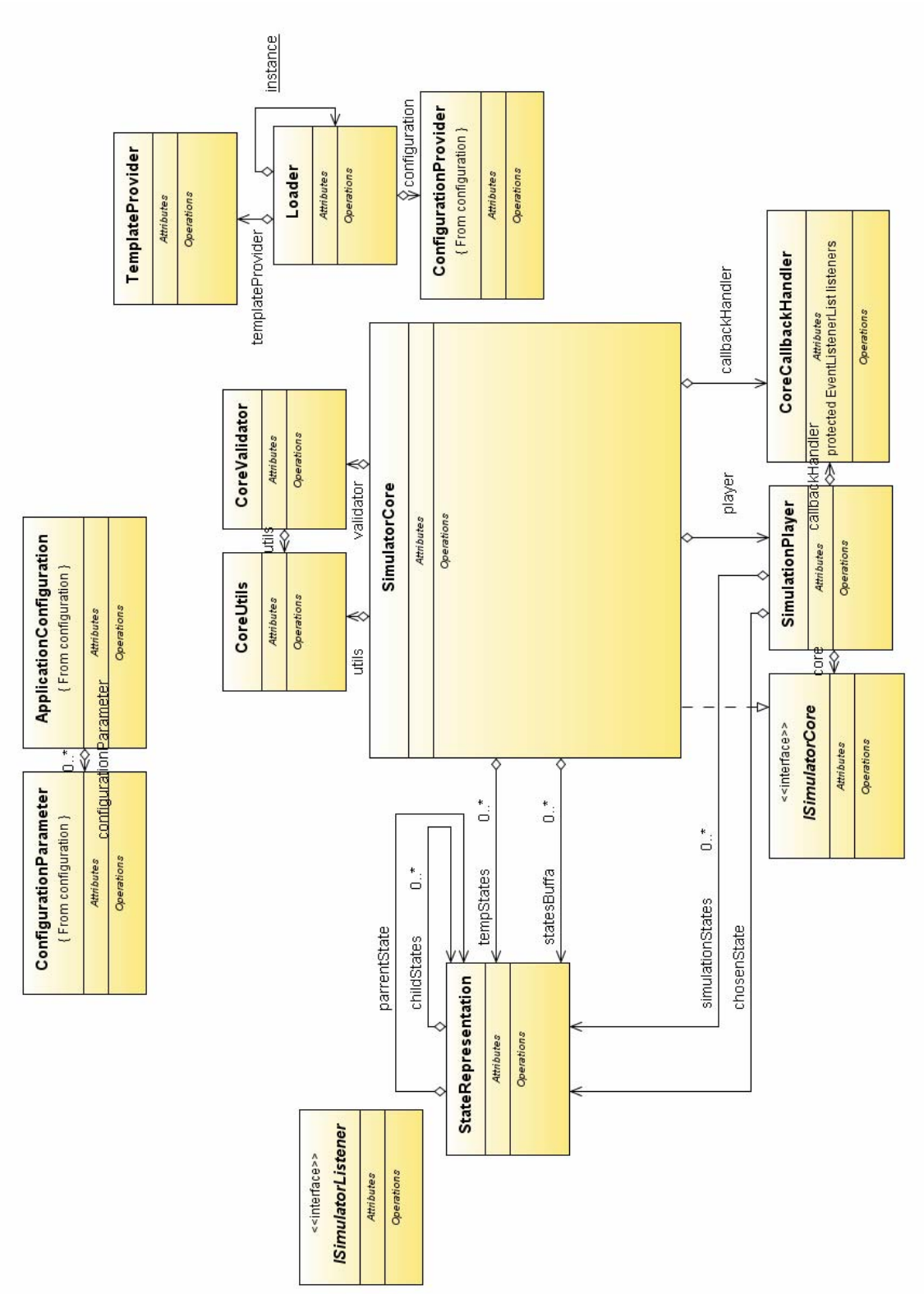
Počas implementácie výslednej aplikácie sa technická stránka v porovnaní s prototypom nezmenila. Aplikácia sa skladá z modulu jadra a z modulov používateľského rozhrania. Jadro poskytuje GUI modulom rozhranie na editáciu a simuláciou automatu. Pri editácií automatu pracuje jadro z triedami, ktoré boli za pomoci knižnice JAXB generované z XML schém. Moduly používateľského rozhrania sú následne o jednotlivých zmenách v automate, či už počas editácie, alebo pri simulácií, informované.

Používateľské rozhranie bolo vytvorené za pomoci knižnice *JUNG*. Aj keď sa počas implementácie objavili problémy týkajúce sa nedostatočnej a chybnej dokumentácie tejto knižnice, jej použitie sa ukázalo ako prínos.

Implementácie prebiehala vo vývojovom prostredí NetBeans.

7.2. DIAGRAM TRIED – JADRO

Táto kapitola obsahuje diagram tried pre modul jadra spolu so stručným opisom ich funkcionality. Diagram je zobrazený na obr.č. 44.

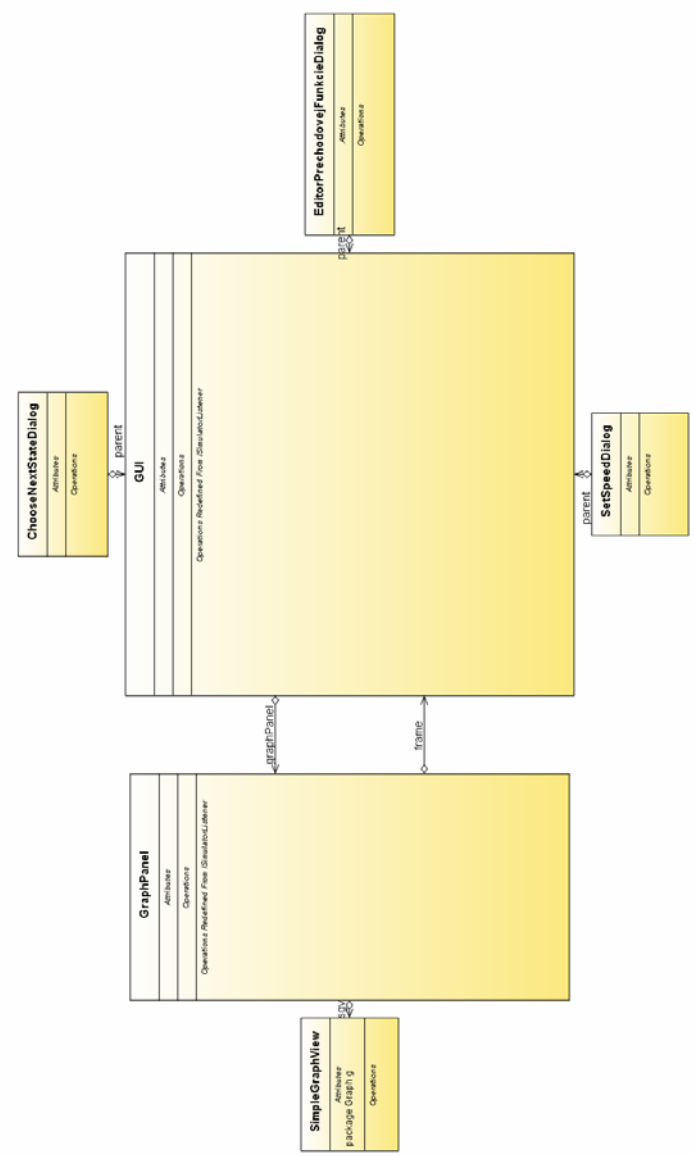


Obr. 44: Diagram tried pre modul jadro

ISimulatorCore je rozhranie, ktoré špecifikuje funkcionality jadra. Rozhranie špecifikuje funkcionalitu, ktorú GUI využíva na editáciu automatu a následné riadenie samotnej simulácie. Toto rozhranie implementuje trieda *SimulatorCore*. *SimulatorCore* je trieda, ktorá obsahuje väčšinu výpočtovej logiky nutnej na editáciu a simuláciu automatu. Obsahuje štruktúry, ktoré reprezentujú automat a logiku, ktorá je schopná tieto štruktúry upravovať podľa požiadaviek GUI. Pre potreby editácie využíva služby triedy *CoreValidator*. Táto trieda validuje jednotlivé entity vo vzťahu k špecifikácií automatu. Pomocou tejto triedy udržuje jadro formálnu správnosť editovaného automatu. Trieda *SimulationPlayer* je jadrom využívaná pri samotnej simulácií. Táto trieda umožňuje riadené prechádzanie po strome jednotlivých stavov, ktorý vytvorilo jadro, a tým simuluje samotné vykonávanie automatu. Podporné činnosti, ako vyhľadávanie v zoznamoch a podobné, zabezpečuje trieda *CoreUtils*. Spätnú komunikácia s GUI modulmi registrovanými v jadre zabezpečuje trieda *CoreCallbackHandler*. Táto trieda je schopná synchrónne a asynchrónne prevolávať jednotlivé moduly GUI na definovanom rozhraní *ISimulatorListener*. Rozhranie *ISimulatorListener* obsahuje metódy, ktoré informujú GUI o zmenách na automate a priebehu simulácie automatu. Programová reprezentácia stavu je uložená v trieda *StateRepresentation*. Informácie o názve stavu, stave zásobníka, pások a pozície hláv na nich sú reprezentované touto triedou. Konfiguračné parametre aplikácie sú spravované triedou *ConfigurationProvider*, ktorej inicializáciu má na starosti trieda *Loader*. Tried *Loader* je trieda zodpovedná za inicializáciu tried, ktorých jediná inštancia je používaná v celej aplikácií. Medzi tieto triedy patrí už spomínaná trieda *ConfigurationProvider,* ale aj trieda *TemplateProvider*. Spravovanie šablón jednotlivých druhov automatov ma na starosti práve trieda *TemplateProvider*.

7.3. DIAGRAM TRIED – GUI

Táto kapitola obsahuje diagram tried pre modul grafického používateľského rozhrania spolu so stručným opisom ich funkcionality. Diagram je zobrazený na obr.č. 45.

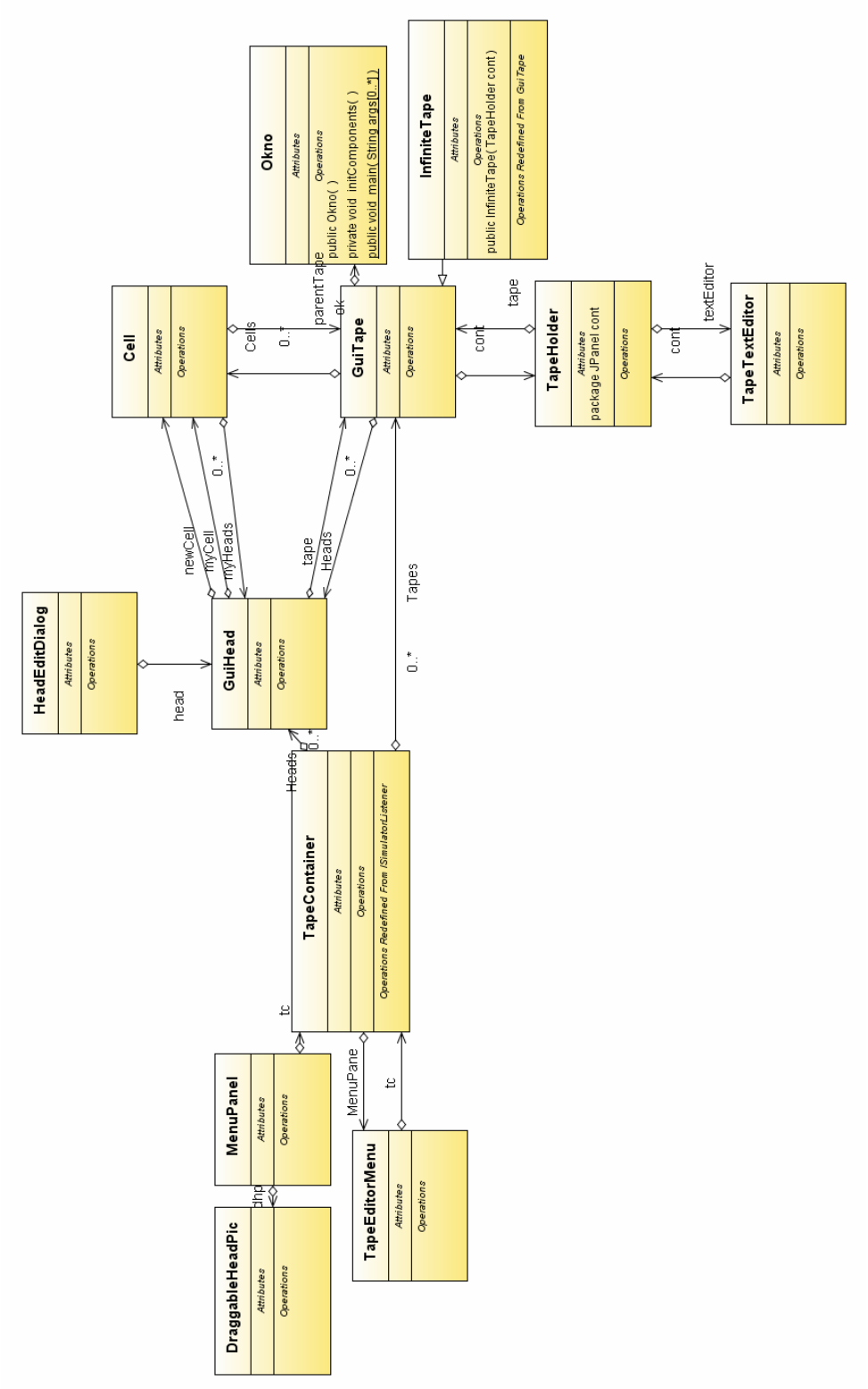


Obr. 45: Diagram tried pre modul GUI

Trieda *GUI* predstavuje hlavné okno aplikácie, stará sa o všetky dôležité úlohy. Je rozdelená na 5 resp. 6 častí. Obsahuje menu, ktoré umožňuje úplnú kontrolu nad aplikáciou. Ďalej obsahuje formálnu špecifikáciu, ktorú je možne editovať. Panel *Tape* zobrazuje pásku automatu a umožňuje jej editáciu. Dialóg *EdititorPrechodovejFunkcie* sa stará o pridanie, respektíve editovanie funkcie, a to priamo v jadre ako aj v grafe zobrazenom v *GraphPanely*. Trieda *SetSpeedDialog* umožňuje grafické nastavenie rýchlosti simulácie. Dialóg *ChooseNextStateDialog* sa zobrazí pri nedeterministickom stave, umožňuje výber nasledujúceho

stavu ako aj grafické zvýraznenie označenej možnosti v grafe. Trieda *GraphPanel* je implementovaná ako JPanel. Táto trieda slúži na zobrazovanie grafu, jeho editovanie (pridávanie vrcholov, prechodových funkcií) a modifikovanie (približovanie, odďaľovanie, zmeny tvaru grafu a pod.). Trieda *SimpleGraphView* predstavuje jednoduchý graf využívaný v *GraphPanely*. *SimulationTreeFrame* umožňuje vytvorenie stromu simulácie, ďalej zobrazuje stavy, ktoré boli alebo neboli akceptované. Umožňuje taktiež uloženie do súboru a transformáciu grafu.

Samostatnou časťou grafického používateľského rozhrania je Páska. Diagram tried, ktorý ju reprezentuje, je zobrazený na obr. č. 46


Okno
Attributes
Operations
public Okno( )
private void initComponents( )
public void main(String args[0..*1])
InfiniteTape
Attributes
Operations
public InfiniteTape(TapeHolder cont)
Operations Redefined From GuiTape
Cell
Attributes
Operations
GuiTape
Attributes
Operations
TapeHolder
Attributes
package JPanel cont
Operations
TapeTextEditor
Attributes
Operations
HeadEditDialog
Attributes
Operations
GuiHead
Attributes
Operations
TapeContainer
Attributes
Operations
Operations Redefined From ISimulatorListener
MenuPanel
Attributes
Operations
DraggableHeadPic
Attributes
Operations
TapeEditorMenu
Attributes
Operations
parentTape
ok
Cells
0..*
cont
tape
textEditor
newCell
myCell
myHeads
tape
Heads
0..*
Tapes
head
Heads
tc
MenuPane
tc
chp

Obr. 46: Diagram tried pre pásku

7.4. VYBRANÉ TRIEDY JADRA

Jadro riadi celý proces vykonávaný v aplikácii.

BALÍK CORE

- ***AsynchronousEventTrigger*** – zabezpečuje, aby nedochádzalo ku konfliktom medzi notifikáciu a grafickým používateľským rozhraním
- ***CoreCallbackHandler*** – trieda, ktorá zabezpečuje volanie jadra na grafickom používateľskom rozhraní. Zaznamenáva zmenu stavu do ktorého automat prešiel, pridanie nového stavu, zmazanie stavu, zmenu stavu na počiatočný alebo koncový stav, pridanie nového prechodu medzi stavmi, zmazanie prechodu, zmenu prechodu, zistí či je automat v koncovom stave a informuje o ukončení simulácie. Zachytí požiadavku na načítanie súboru, požiadavku na vytvorenie prázdneho automatu. Oznamuje nutnosť vybratia nasledujúceho stavu pri nedeterminizme v interaktívnom móde. Zaznamenáva aj pridanie a zmazanie hlavy a pásky, pozíciu pridanej aj zmazanej hlavy, zmenu obsahu pásky, zmenu polohy pásky. Zistí aj pracovný mód – či vyberá nasledujúci krok automaticky alebo sa spýta používateľa.
- ***CoreUtils*** – obsahuje pomocnú funkcionalitu pre jadro. Pracuje napríklad s páskami, hlavami, zásobníkom...
- ***CoreValidator*** – kontroluje, či sú všetky objekty správne zadefinované, v správnom stave a či sú umožnené správne operácie
- ***SimulatorCore*** – implementácia jadra simulátora. Pracuje so všetkými komponentmi (s páskami, stavmi, prechodmi, testovacím módom, abecedou...)
- ***ISimulatorCore*** – rozhranie, ktoré definuje funkcionalitu, ktorú poskytuje jadro
- ***SimulatorCoreException*** – výnimka informujúca o chybe v jadre
- ***SimulatorEvent*** – udalosti jadra simulátora
- ***ISimulatorListener*** – rozhranie, ktoré definuje udalosti, ktoré môže jadro volať. Následne ich používateľské prostredie spracuje
- ***Loader*** – inicializuje a následne poskytuje inštancie pre objekty, pre ktoré existuje len jedna inštancia na celý program, napr. ApplicationContext spring frameworku, a Logger

- ***SimulationPlayer*** – spravuje prehliadanie simulácie. Kontroluje či simulátor skončil prehliadanie simulácie. Nastaví či sa má krokovať simulácia, alebo prebehne do konca, či je v automatickom alebo interaktívnom režime. Umožňuje pozastavenie, pokračovanie alebo prehratie od začiatku simulácie.
- ***StateRepresentation*** – reprezentácia stavu. Zaoberá sa stavom hláv, pások, stavu, predchádzajúcim stavom, zaradením stavu v strome stavov a zásobníkom.
- ***TemplateProvider*** – poskytuje šablóny jednotlivých konfigurácii typov automatov za pomoci spring frameworku.

BALÍK CORE.STATEMACHINE

Obsahuje jaxb triedy vygenerované zo stateMachine.xsd, ktoré slúži ako šablóna pre konfiguráciu jednotlivých parametrov.

BALÍK CORE.TEMPLATE

Obsahuje jaxb triedy vygenerované z Template.xsd

7.5. XML SCHÉMY

XML schémy slúžia na definíciu vlastností simulátora. Sú v nich uložené parametre simulátora, ktoré sú deserializované za využitia knižnice JAXB. To má na starosti balík configuration.

BALÍK CONFIGURATION

Tento balíček plní funkciu konfigurácie všetkých častí aplikácie v závislosti od typu automatu. Keďže každý typ automatu má svoje špecifické vlastnosti, v tomto balíčku sa z príslušného XML súboru aplikujú dané vlastnosti na vybraný typ automatu.

- ***ApplicationConfiguration*** – je jaxb trieda vygenerovaná z Configuration.xsd a slúži ako šablóna pre konfiguráciu simulátora
- ***ConfigurationParameter*** – nastavenie hodnôt parametrov
- ***ConfigurationProvider*** – konfigurácia rôznych vlastností, ktoré sú nemenné počas simulácie ako je jazyk, zobrazenie prázdneho znaku, maximálna hĺbka stromu pri nedeterminizme, zobrazenie prázdneho symbolu na páske
- ***ObjectFactory*** – povoľuje konštruovať nové inštancie Java reprezentácie XML obsahu

7.6. VYBRANÉ TRIEDY GUI

Oproti prototypu zo zimného semestra sa vytvorilo sa nové lištové menu, ktoré už zahŕňa všetky prvky na ovládanie funkcionality, pričom niektoré z nich, tak ako predtým, sú pre ľahšiu dostupnosť spracované aj graficky v príslušných oknách simulátora. Pre jednoduchšiu manipuláciu sú to hlavne prvky na ovládanie simulácie a na tvorbu stavových diagramov pri určených typoch automatov. Nová je aj grafická reprezentácia pásky.

Na zobrazovanie stavových diagramov sme sa rozhodli použiť javovskú knižnicu JUNG.

BALÍK PROTOTYPE GUI

Tento balík zodpovedá za grafickú reprezentáciu.

- ***ChooseNextStateDialog*** – dialógové okno na výber nasledujúceho kroku používateľom pri nedeterminizme automatu v zapnutom interaktívnom móde
- ***Constants*** – definuje cesty k obrázkom, ktoré reprezentujú v stavovom diagrame vstupný stav, normálny stav, konečný stav a ich zvýraznenie ako aktuálneho stavu počas simulácie
- ***Edge*** – táto trieda predstavuje hranu medzi dvoma uzlami (stavmi)
- ***EditorPrechodovejFunkcieDialog*** – umožňuje zmenu alebo pridanie prechodových funkcií
- ***GUI*** - jednoduché grafické rozhranie pre potreby prototypu. Je rozdelené na
- päť hlavných častí a to: menu, formálna špecifikácia, editor stavového diagramu, editor vstupnej pásky a ovládací panel simulácie)
- ***GraphPanel*** – v tomto paneli sa vykresľuje graf. Tento panel ma možnosť priblíženia a oddialenia grafu, ako aj posúvania grafu, obsahuje metódy na skopíruje viditeľnej časti grafu do súborov png alebo jpeg a umožňuje zmenu ikon vrcholov grafu.
- ***SetSpeedDialog*** – nestavenie rýchlosti simulácie
- ***SimpleGraphView*** – slúži na vykreslenie grafu
- ***SimulationTreeFrame*** – vykresľuje strom všetkých možných ciest pre daný vstup na páske

BALÍK PROTOTYPE GUI.SMALL

Slúži na vykreslenie dodatočných panelov v editore prechodovej funkcie.

BALÍK GUI.PASKA

Grafická reprezentácia vstupnej pásky

- ***Cell*** – načítanie vstupu, buď ako reťazca znakov do textového poľa, alebo zadávanie po jednom znaku do jednotlivých buniek pásky
- ***CellDragHandler*** – pridanie bunky do pásky
- ***DraggableHeadPic*** – vloženie hlavy na pásku systémom drag and drop
- ***GuiHead*** – nastavenie parametrov pásky ako číslo pásky, aktuálnu pozíciu čítaného symbolu, obsahuje metódy na menenie nastavenia pohybu hlavy, čítanie a zapisovanie na pásky
- ***GuiTape*** - grafická reprezentácia pásky. Nastavuje maximálny počet buniek, ktoré sa zobrazia vo výseku pásky, editácia pásky, pohyb po páske pomocou smerových tlačidiel, pridanie novej bunky do pásky
- ***HeadEditDialog*** – okno pre možnosť nastavenie pásky aj na zápis a pohyb smerom doľava
- ***InfiniteTape*** – reprezentácia „nekonečnej " pásky, zobrazí výsek maximálneho počtu buniek, pri posune doľava alebo doprava pomocou smerových tlačidiel pridáva bunky so symbolom Blank
- ***TapeConfiguration*** – konfigurácia parametrov pásky ako nastavenie prázdneho symbolu, maximálny počet zobrazovaných buniek, rozmery bunky, rýchlosť pohybu hlavy po páske, povolenie zápisu a pohybu vľavo pre hlavu
- ***TapeContainer*** – okno na vykreslnie komponentov pásky, metódy na prepojenie funkcionality pásky s jadrom
- ***TapeEditorMenu*** – pridanie ďalšej pásky
- ***TapeTextEditor*** – dialógové okno, kde používateľ môže vložiť naraz celý reťazec vstupu, editovať tento reťazec a zmazať pásku

BALÍK PROTOTYPE ICON

Tento adresár obsahuje obrázky, použité na grafické stvárnenie ovládacích prvkov používateľského rozhrania

BALÍK GUI.IMAGES

Tento adresár obsahuje obrázky pre interaktívne prvky na páske.

BALÍK GUI.STACK

Grafická reprezentácia zásobníka

- ***CellStackPanel*** – vytvorenie a naplnenie bunky zásobníka
- ***ColumnStackPanel*** – vytvorenie zásobníka a naplnenie bunkami
- ***StackPanel*** – samostatný panel pre vkladanie zásobníkov
- ***Zasobnik –*** vytvorenie okna pre prácu so zásobníkom

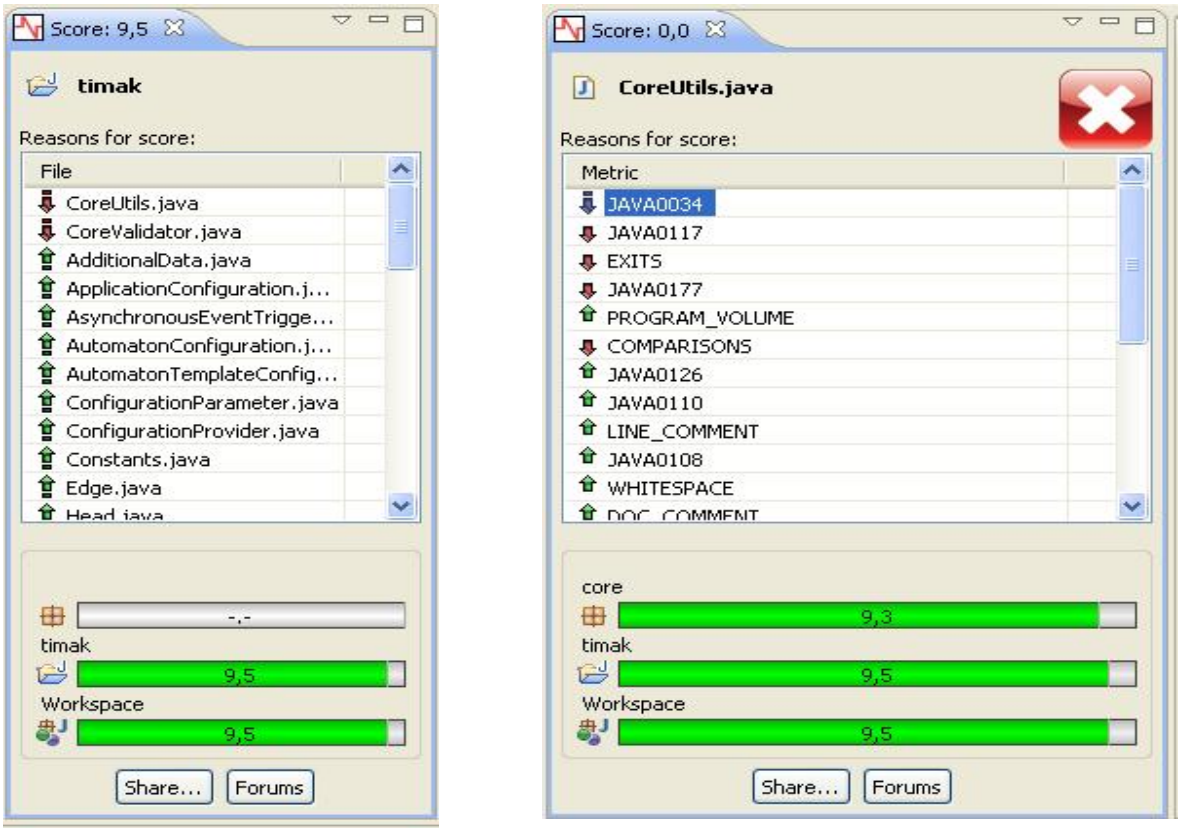
BALÍK JUNG

Pridávanie a editácia vrcholov, hrán, popisov hrán, popisy vrcholov, zmena veľkosti vrcholu

- ***DefaultEdgeLAbelRendererTP*** – editovanie vrcholov a hrán
- ***EdgeFactory*** – pridanie popisu hrany získaný z dialógového okna
- ***EditingGraphMousePluginTP –*** vytváranie grafu, pridávanie nových stavov a hrán
- ***EditingModalGraphMouseTP*** – transformácia, editácia, presun grafu pomocou klávesových skratiek (klávesa t – transformácia, e – editácia, p – presúvanie grafu)
- ***EditingPopupGraphMousePluginTP*** - plugin, ktorý používa popup menu na vytvorenie stavov, a hrán.
- ***LabelEditingGraphMousePluginTP –*** zmena názvov existujúcich vrcholov a hrán po kliknutí na príslušný vrchol
- ***ToStringLabellerTP -*** –transformácia vrcholu na reťazec
- ***VertexShapeSizeAspectTP*** – zmena veľkosti vrcholu

7.7. ENERJY

Energy je Eclipse modul na statickú analýzu java kódu. Pomocou metrík ohodnocuje kódy, na základe ktorých vypočíta ich hodnotenie. Metriky, ktoré slúžia na ohodnocovanie zahŕňajú napríklad počet riadkov java kódu, okomentovanie všetkých metód, a pod. Stupnica hodnotenia je od nula do desať. Náš projekt dosiahol pekné hodnotenie, 9,5.



Obr. 47: Obrazovky výsledku testu pomocou Enerjy

Na obrázku vľavo je modulové okno Enerjy, kde prezentuje svoje výsledky analýzy kódu. Hodnotenie je vypočítané pre celý projekt. V hornej časti je zoznam súborov, ktoré uspešne alebo neúspešne prešli analýzou. V dolnej časti sú tri hodnoty. V poradi zvrchu nadol je ohodnotenie balíka, projektu a nakoniec celého workspace.

Na obrázku vpravo je hodnotenie súboru; v hornej časti sú znovu farebne označené úspešné a neúspešné výsledky jednotlivych metrík v danom súbore.

8. TESTOVANIE

Poslednou fázou vývoja aplikácie bolo testovanie. Jeho cieľom bolo overenie funkcionality kľúčových častí systému a zároveň určiť reálnu mieru použiteľnosti aplikácie na simuláciu výpočtových zariadení. Keďže sme ako techniku vývoja zvolili agilné programovanie, aplikáciu sme testovali už počas vývoja, a tak sme včas odhalili viacero chýb. Na ich zaznamenávanie sme použili systém na sledovanie chýb Mantis. Na overenie funkcionality výslednej aplikácie sme zvolili metódu testovania formou čiernej skrinky. Testy vykonávali primárne všetci členovia tímu. Aplikáciu sme dali otestovať aj niekoľkým študentom mimo nášho tímu, ktorí v minulosti absolvovali predmet Teoretické základy informatiky, a teda boli oboznámení s problematikou automatov.

Testovali sme voči definovaným testovacím prípadom. Základné testovacie prípady sme definovali už pri testovaní prototypu a boli samozrejme doplnené o overovanie novo pridanej funkcionality.

Po dôkladnom otestovaní môžeme skonštatovať, že aplikácie simuluje správanie sa výpočtových zariadení správne a neobsahuje významné chyby. Aj študenti mimo nášho tímu, ktorí aplikáciu stručne testovali, považujú aplikáciu za funkčnú a prínosnú pri študovaní problematiky automatov.

TC_ID	Testovací prípad
Popis	Názov, stručný popis testovacieho prípadu.
Nastavenie	Nastavenie systému do počiatočného stavu, aby bolo možné vykonať test.
Vykonané kroky	Postupnosť krokov testovacieho prípadu.
Očakávaný výsledok	Popis očakávaného výsledku testu.
Stav podmienky	Vyhovuje – výsledok testovania zodpovedá očakávanému výsledku

Tab. 7: Vysvetlivky k prípadom použitia

TC01 – Načítanie vstupného súboru

TC_ID	TC01
Popis	Načítanie vstupného súboru
Nastavenie	Spustenie aplikácie
Vykonané kroky	1) Kliknúť na File -> Open. 2) Vybrať súbor *.xml, kliknúť na Open.
Očakávaný výsledok	• Zobrazí sa stavový diagram reprezentujúci načítaný automat. • Zobrazí sa formálna špecifikácia načítaného automatu. • Zobrazí sa vstupná páska.
Stav podmienky	Vyhovuje

TC02 – Krokovanie simulácie, deterministický krok

TC_ID	TC02
Popis	Krokovanie simulácie, deterministický krok
Nastavenie	Načítanie vstupného súboru
Vykonané kroky	1) Stlačiť tlačidlo “krok”.
Očakávaný výsledok	• Stav automatu sa zmení podľa príslušnej prechodovej funkcie. • Čítacia hlava sa posunie na ďalší symbol.
Stav podmienky	Vyhovuje

TC03 – Krokovanie simulácie, nedeterministický krok

TC_ID	TC03
Popis	Krokovanie simulácie, nedeterministický krok
Nastavenie	Načítanie vstupného súboru, krokovanie simulácie tak, aby bol nasledujúci krok nedeterministický
Vykonané kroky	1) Stlačiť tlačidlo “krok”.
Očakávaný výsledok	• Zobrazí sa dialog pre výber nového stavu. Ponúknuté stavy budú zodpovedať prechodovým funkciám pre aktuálny stav a vstupný symbol.

Stav podmienky	Vyhovuje

TC04 – Výber stavu pri nedeterministickom kroku

TC_ID	TC04
Popis	Výber stavu pri nedeterministickom kroku
Nastavenie	Načítanie vstupného súboru, stlačenie tlačidla "krok" pri nedeterministickom kroku
Vykonané kroky	1) Vybrať jeden z ponúknutých stavov. 2) Stlačiť OK.
Očakávaný výsledok	• Stav automatu sa zmení na vybraný stav. • Čítacia hlava sa posunie na ďalší symbol.
Stav podmienky	Vyhovuje

TC05 – Spustenie simulácie do konca

TC_ID	TC05
Popis	Spustenie simulácie do konca
Nastavenie	Načítanie vstupného súboru
Vykonané kroky	1) Stlačiť tlačidlo "play".
Očakávaný výsledok	• Simulácia prebehne do konca. • Ak existuje aspoň jedno správne riešenie, zobrazí sa postupnosť prechodov pre toto riešenie. • Používateľ má možnosť pokračovať v hľadaní ďalších riešení.
Stav podmienky	Vyhovuje

TC06 – Resetovanie simulácie

TC_ID	TC06
Popis	Resetovanie simulácie
Nastavenie	Načítanie vstupného súboru, priebeh simulácie krokovaním alebo dokonca.
Vykonané kroky	1) Stlačiť tlačidlo "stop".

Očakávaný výsledok	• Stav automatu sa zmení na počiatočný stav. • Čítacia hlava sa presunie na začiatok vstupu.
Stav podmienky	Vyhovuje

TC07 – Voľba rýchlosti simulácie

TC_ID	TC07
Popis	Voľba rýchlosti simulácie
Nastavenie	Spustenie aplikácie
Vykonané kroky	1) Kliknúť na želanú úroveň rýchlosti na posuvnom ukazovateli. 2) Kliknúť na Simulation -> Simulation settings -> Simulation speed
Očakávaný výsledok	• Nastaví sa želaná rýchlosť simulácie.
Stav podmienky	Vyhovuje

TC08 – Pridanie pásky

TC_ID	TC08
Popis	Pridanie pásky
Nastavenie	Vytvorenie automatu
Vykonané kroky	1) Kliknúť na „Add tape“.
Očakávaný výsledok	• Do plátna pre pásku je pridaná nová páska. (Musí byť možné pridať viac pások naraz)
Stav podmienky	Vyhovuje

TC09 – Editovanie pásky

TC_ID	TC09
Popis	Editovanie pásky
Nastavenie	Páska je vytvorená
Vykonané kroky	1) Kliknúť na tlačidlo editovať pásku („ceruzka“). 2) Kliknúť na tlačidlo pridať nové pole pásky („+“).

Očakávaný výsledok	• Je možné zadať obsah pásky (buď celej naraz alebo samostatne po poliach).
Stav podmienky	Vyhovuje

TC10 – Vymazanie pásky

TC_ID	TC10
Popis	Vymazanie pásky
Nastavenie	Páska je vytvorená
Vykonané kroky	1) Kliknúť na tlačidlo editovať pásku („ceruzka“). 2) Kliknúť na „Delete tape“.
Očakávaný výsledok	• Páska je vymazaná.
Stav podmienky	Vyhovuje

TC11– Pridanie hlavy na pásku

TC_ID	TC11
Popis	Pridanie hlavy na pásku
Nastavenie	Páska je vytvorená, má zadané polia
Vykonané kroky	1) Držať stlačené tlačidlo pridať hlavu („symbol hlavy“) a preniesť ju na želané miesto pásky.
Očakávaný výsledok	• Hlava je vytvorená na želanom mieste a je označená správnym číslom (musí byť možné pridať viac hláv naraz).
Stav podmienky	Vyhovuje

TC12 – Zmena hlavy na páske

TC_ID	TC12
Popis	Zmena hlavy na páske
Nastavenie	Hlava je umiestnená na páske
Vykonané kroky	1) Dvakrát kliknúť na hlavu.
Očakávaný výsledok	• Otvorí sa okno „Edit tape properties“, kde je možné meniť vlastnosti hlavy.

Stav podmienky	Vyhovuje

TC13 – Vymazanie hlavy z pásky

TC_ID	TC13
Popis	Vymazanie hlavy z pásky
Nastavenie	Hlava je umiestnená na páske
Vykonané kroky	1) Dvakrát kliknúť na hlavu. 2) Kliknúť na „Delete tape“.
Očakávaný výsledok	• Hlava je vymazaná.
Stav podmienky	Vyhovuje

TC14 – Pridanie stavu

TC_ID	TC14
Popis	Pridanie stavu
Nastavenie	Vytvorenie automatu
Vykonané kroky	a) 1) Kliknúť na plátno pri zvolenom móde „Editing Graph Mode“. b) 1) Kliknúť na tlačidlo „Add State“. 2) Vyplniť názov stavu.
Očakávaný výsledok	• Stav je pridaný na plátne.
Stav podmienky	Vyhovuje

TC15 – Označenie stavu ako začiatočný/koncový

TC_ID	TC15
Popis	Označenie stavu ako začiatočný/koncový
Nastavenie	Stav je vytvorený
Vykonané kroky	1) Kliknúť pravým tlačidlom na stav. 2) Kliknúť na možnosť „Change to End State“/ „Change to Initial State“

Očakávaný výsledok	• Stav je zmenený, výsledok je indikovaný zmenenou ikonou.
Stav podmienky	Vyhovuje

TC16 – Vymazanie stavu

TC_ID	TC16
Popis	Vymazanie stavu
Nastavenie	Stav je vytvorený
Vykonané kroky	1) Kliknúť pravým tlačidlom na stav v móde „Picking Graph Mode“. 2) Kliknúť na možnosť „Delete State“.
Očakávaný výsledok	• Stav je vymazaný, spolu s ním aj prechod v grafe a prechodová funkcia.
Stav podmienky	Vyhovuje

TC17– Pridanie prechodu

TC_ID	TC17
Popis	Pridanie prechodu
Nastavenie	Vytvorenie automatu, sú vytvorené aspoň 2 stavy.
Vykonané kroky	a) 1) Kliknúť na stav pri zvolenom móde „Editing Graph Mode“ a pretiahnuť na druhý stav. 2) Vyplniť hodnotu hrany. b) 1) Kliknúť na tlačidlo „Transition function editor“. 2) Zadať požadované hodnoty.
Očakávaný výsledok	• Prechod je pridaný. Pokiaľ je vytvorená páska s hlavou, prechod je zaznamenaná v tabuľke prechodových funkcií.
Stav podmienky	Vyhovuje

TC18 – Vymazanie prechodu

TC_ID	TC18
Popis	Vymazanie prechodu
Nastavenie	Prechod je vytvorený.
Vykonané kroky	1) Kliknúť pravým tlačidlom na prechod v móde „Picking Graph Mode“. 2) Kliknúť na „Delete Edge“.
Očakávaný výsledok	• Prechod je vymazaný. Spolu s ním je vymazaný aj riadok v prechodovej funkcii.
Stav podmienky	Vyhovuje

9. ZHODNOTENIE

V tejto záverečnej kapitole by sme chceli stručne zhrnúť dosiahnuté výsledky a prínos, ktorý mal Tímový projekt pre náš ďalší profesionálny rast.

Naša práca na tímovom projekte prebiehala v dvoch semestroch. Počas prvého semestra sme sa zamerali na analýzu problematiky a návrh riešenia, pričom hlavným výstupom bol funkčný prototyp. Ten bol schopný v základnej miere odsimulovať jednoduchý konečný automat, ukázal správnosť zvolených prístupov, technológií a algoritmov a položil tak základný kameň pred implementáciou plnohodnotného simulátora.

Tá prebiehala počas celého letného semestra. Zároveň sme doplnili analýzu a návrh riešenia o 2 nové typy výpočtových zariadení – RAM stroj a počítadlový stroj. Pri implementácii sme sa zamerali na najdôležitejšie aspekty simulátora opísané v návrhu riešenia. Keďže náš tím v priebehu semestra opustil jeden člen, nepodarilo sa nám implementovať doplnkovú funkcionalitu opísanú v návrhu. Ako celok však vytvorená aplikácia predstavuje plnohodnotný simulátor výpočtových zariadení.

Keďže zadanie nášho tímového projektu bolo pomerne rozsiahle, strávili sme prácou na ňom viac času, než je pre tento predmet pridelené. Na druhej strane sme však získali rozsiahle skúsenosti s prácou v tíme, čo bude mať nepochybne veľký prínos do našej ďalšej praxe. Zároveň sme mali príležitosť oboznámiť sa s novými technológiami a vyskúšať plnohodnotný vývoj softvéru, kde sme zúročili mnoho doteraz nadobudnutých vedomostí.

10. LITERATÚRA

[1] M. Nehéz, D. Chudá, I. Polický, M. Čerňanský: Teoretické základy informatiky, september 2006

PRÍLOHA A – POUŽÍVATEĽSKÁ PRÍRUČKA

Táto kapitola opisuje prácu s celým simulátorom. Obsahuje opis aplikácie, softvérové a harvérové požiadavky pre jej správnu funkčnosť a návod k použitiu jednotlivých častí aplikácie. Keďže simulátor je určený pre študentov informatiky, predpokladá sa znalosť práce s podobnými aplikáciami.

Softvérové požiadavky:

- MS Windows XP/Vista
- JDK 1.6.0_13
- JRE 6

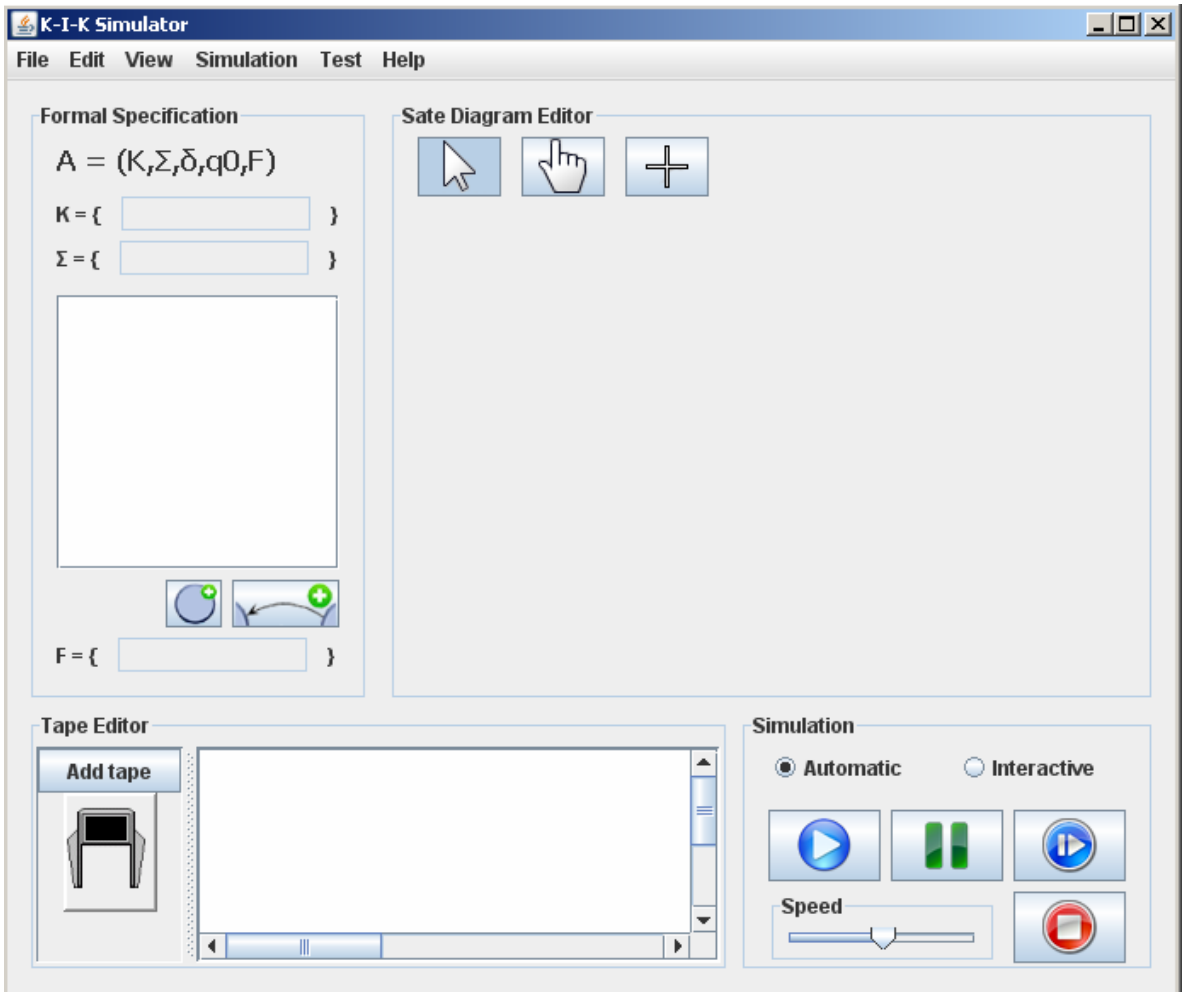
Hardvérové požiadavky:

- 512 MB RAM
- Rozlíšenie displeja aspoň 800 x 600

10.1. SPUSTENIE APLIKÁCIE

Otvorte príkazový riadok: Start → Run, tu napíšte `cmd`.
Do príkazového riadku napíšte `java -jar Simulator.jar` .

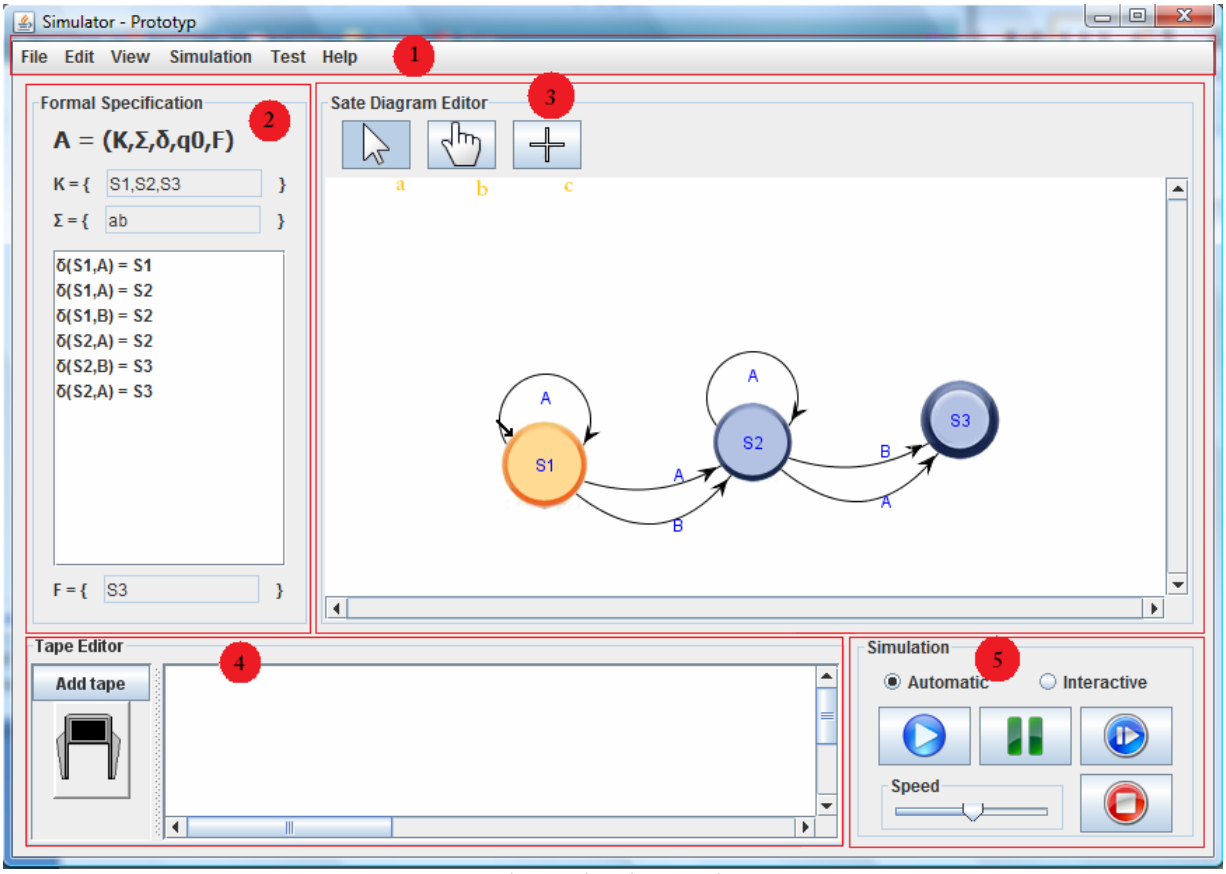
Po úspešnom spustení sa zobrazí hlavné okno, ktoré vyzerá nasledovne.



Obr. 48: Hlavné okno aplikácie

Pri každom type automatu vyzerá hlavné okno trochu odlišne. Jednotlivé odlišnosti sú zobrazené a opísané v nasledujúcich podkapitolách.

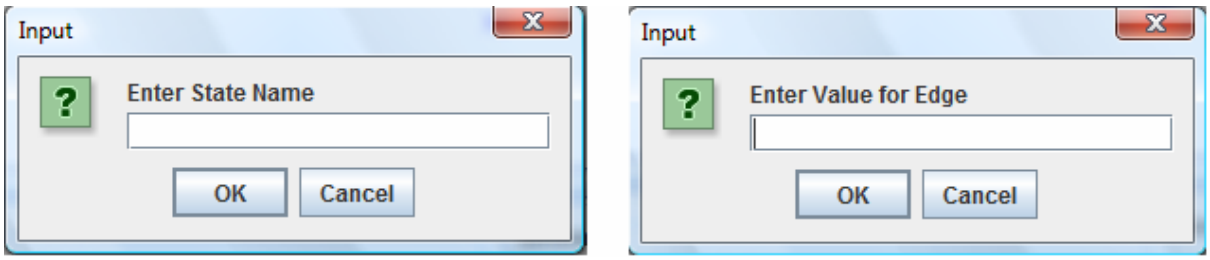
KONEČNÝ AUTOMAT



Obr. 49: Hlavné okno konečného automatu

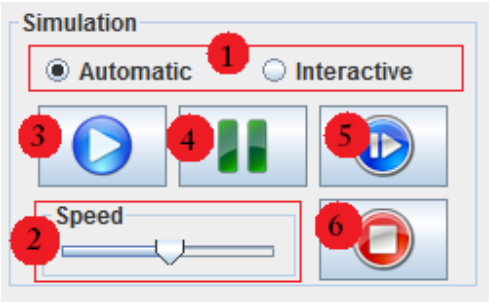
1. **Menu aplikácie** – rovnaké pre všetky typy automatov
2. **Formálna špecifikácia automatu** – obsahuje množinu stavov K, abecedu Σ, zoznam funkcií, začiatočný stav a množinu koncových stavov
3. **Diagram stavového automatu** – obsahuje panel na kreslenie automatov a 3 ikony:
   a. *Šípka* – presúvanie celého automatu. Pre posun grafu v GUI ho jednoducho ľavým tlačidlom na myške označíte, držíte a presuniete na zvolené miesto. Ak chcete graf priblížiť alebo oddialiť, použite koliesko na myške.
   b. *Ruka* – označenie jedného objektu (stavu, prechodu) kliknutím naň, alebo označenie viacerých stavov nakreslením virtuálneho obdĺžnika
   c. *Plus* – pridávanie stavov kliknutím na plochu, alebo pridávanie prechodových hrán kliknutím na začiatočný stav a potiahnutím virtuálnej čiary do koncového stavu

Po pridaní hrany je potrebné zadať hodnotu hrany a po pridaní stavu je potrebné zadať meno nového stavu. Obrazovky sú zobrazené na obr. 3



Obr. 50: Vloženie mena pre stav (vľavo) a hodnoty hrany (vpravo)

4. **Editor pások** – Konečné pásky s hlavami. Editor obsahuje viacero tlačítok:
   a. *Ceruzka* – editovanie obsahu pásky
   b. *Plus* – pridanie bunky do pásky
   c. *Dvojité šípky* – posun pásky doľava alebo doprava
   d. *Hlava* – pridanie hlavy na pásku pretiahnutím ikony na miesto kde ju chceme, alebo napísaním čísla pásky, ak ich je viac
   e. *Pásky* – pridanie novej prázdnej pásky
5. **Simulácia** – Oblasť viacerých tlačidiel, ktorými sa riadi simulácia (obr.38):
   a. *Voľba pri nedeterminizme* – automatický výber správnej cesty (Automatic), alebo spýtanie sa používateľa na ďalšiu cestu (Interactive)
   b. *Rýchlosť* – pomocou „slideru“ je možné vybrať rýchlosť simulovania (od do)
   c. *Štart* – spustenie simulácie
   d. *Pauza* – pozastavenie simulácie, je možné v nej ďalej pokračovať
   e. *Krokovanie* – prechádzanie automatu po jednotlivých krokoch
   f. *Zastavenie* – simulácia sa zastaví a vráti sa do začiatočného stavu

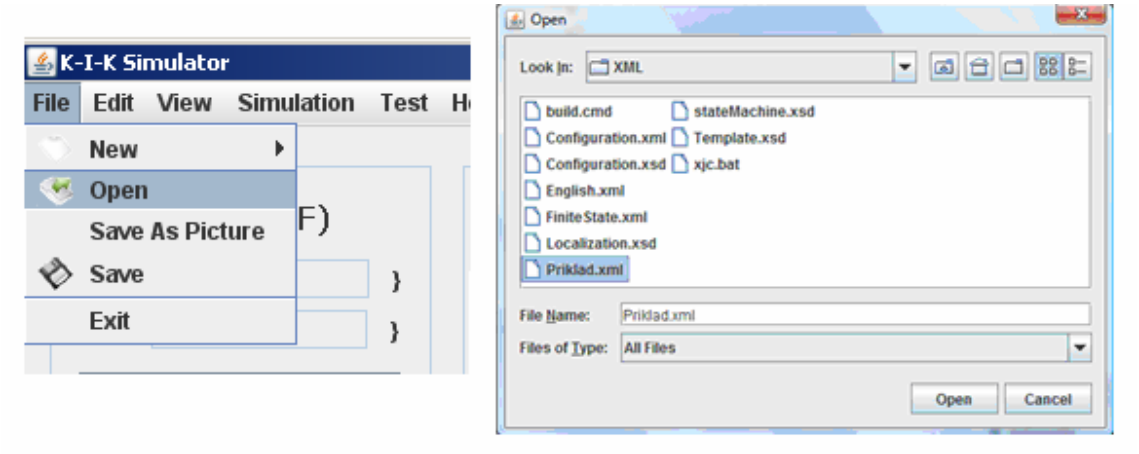


2

Obr. 51: Riadenie simulácie

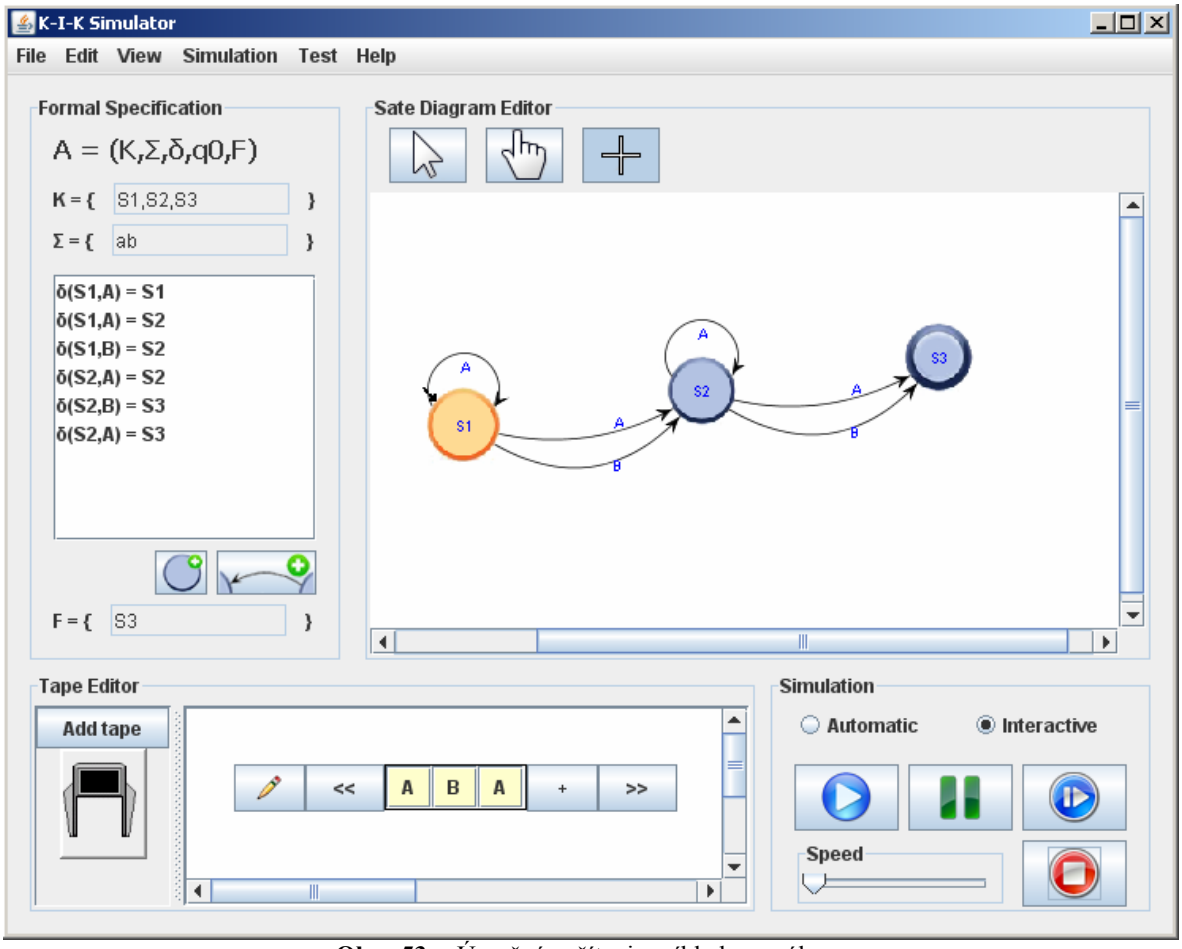
A.2. NAČÍTANIE PRÍKLADU ZO SÚBORU

Ak chcete načítať XML súbor, kliknete na File → Open. Následne sa zobrazí okno, kde si vyberiete XML súbor (príklad), ktorý chcete načítať.



Obr. 52: Otvorenie súboru

Po úspešnom načítaní sa zobrazí správa o úspešnom načítaní a v hlavnom menu sa vykreslia jednotlivé stavy, prechodové funkcie, graf a vstupná páska.



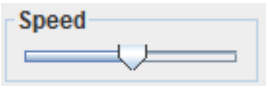
Obr. 53: : Úspešné načítanie príkladu zo súboru

A.3. SPUSTENIE EDITORA NA TVORBU VLASTNÉHO DIAGRAMU

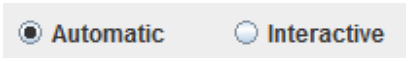
A.4. SPUSTENIE SIMULÁCIE

Nastavenia pred spustením:

Pomocou posuvníka môžte zvoliť rýchlosť simulácie. Rozsah je 0 až 2 sekundy.



Pre zvolenie spôsobu simulácie pri nedeterminizme zvoľte jednu z dvoch možností:



- Automatic – simulátor vyberie sám správnu cestu
- Interactive – používateľ pri nedeterministickom kroku musí zvoliť stav, do ktorého má ísť

Na spustenie simulácie vyberte jednu z nasledujúcich možností:

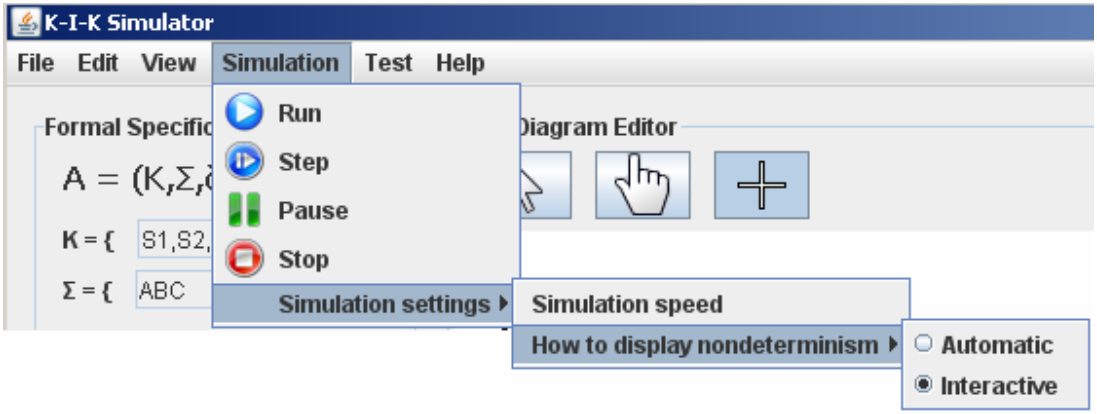
3. Ak chcete spustiť simulovanie bez zastavenia, stlačte tlačidlo „Run“.

4. Ak máte záujem o simuláciu prostredníctvom jednotlivých krokov, stlačte pre každý nasledujúci krok tlačidlo „Step“.

Pre pozastavenie simulácie stlačte tlačidlo „Pause“.

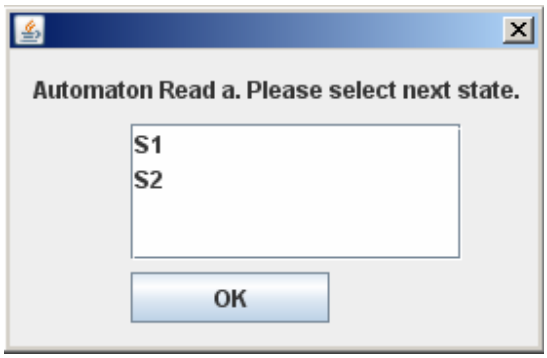
Ak chcete reštartovať simuláciu, stlačte tlačidlo „Stop“.

Predchádzajúce činnosti môžte riadiť aj pomocou horného menu:



A.5. VÝBER NASLEDUJÚCEHO KROKU

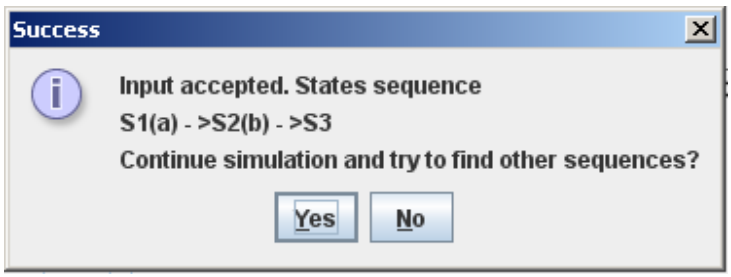
Ak existujú viaceré možnosti na prechod do ďalšieho stavu, zobrazí sa okno, v ktorom používateľ určí nasledujúci stav.



Obr. 54: Výber nasledujúceho kroku

A.6. AKCEPTOVANIE

Ak simulátor akceptuje vstup, tak prostredníctvom dialógového okna túto skutočnosť oznámi používateľovi. Taktiež mu ponúkne možnosť pokračovať v simulácii a hľadať inú sekvenciu krokov.



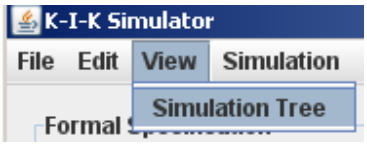
Obr. 55: Akceptovanie vstupu automatom

A.7. NEAKCEPTOVANIE

Obr. 56: Neakceptovanie vstupu automatom

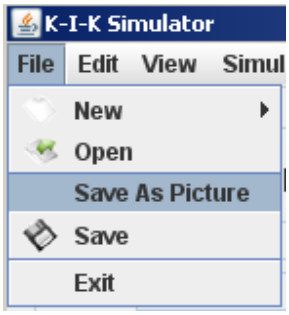
A.8. ZOBRAZENIE SIMULAČNÉHO STROMU

Priebeh simulácie si možno po jej skončení pozrieť v simulačnom strome, ktorý prehľadne zobrazuje postupnosť vykonaných krokov a ostatné možnosti pri nedeterministickom automate.

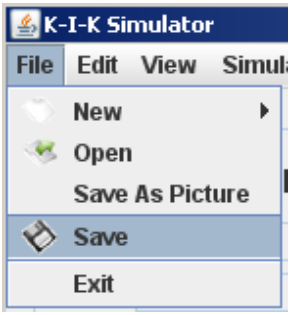


A.9. UKLADANIE

Diagram ako obrázok:



Uloženie ako XML dokument:


K-I-K Simulator
File
Edit
View
Simul
New
Open
Save As Picture
Save
Exit