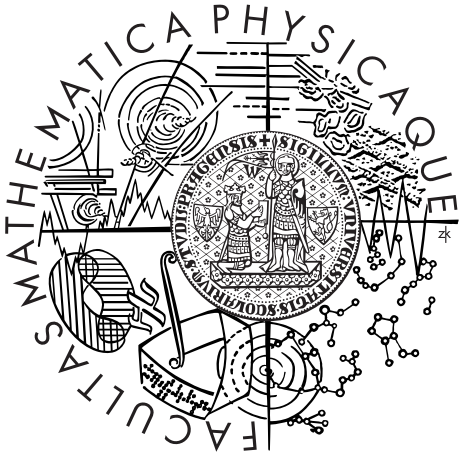
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jiří Vytasil

Univerzální diskrétní simulátor

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Martin Pergel, Ph.D.
Studijní program: Informatika
Studijní obor: Správa počítačových systémů

Praha 2013

Rád bych poděkoval mému vedoucímu bakalářské práce, panu RNDr. Martinovi Pergelovi, Ph.D., za ochotu, cenné rady a vstřícnost, které mi projevoval v průběhu psaní programu a této práce. Dále bych rád poděkoval svým kolegům, se kterými jsem mohl konzultovat technologie a postupy. Nakonec bych rád poděkoval své rodině za jejich podporu.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 20. 5. 2013 Podpis autora

Název práce: Univerzální diskrétní simulátor

Autor: Jiří Vytasil

Katedra: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Martin Pergel, Ph.D., Kabinet software a výuky informatiky

Abstrakt: V této práci je rozebrán návrh a implementace programu umožňujícího provádět diskrétní simulace. Dále je v programu implementována vizualizace průběhu diskrétní simulace. Důraz je kladen především na univerzalitu. Kromě programu a jeho popisu jsou zde informace o diskrétní simulaci, které umožní jednodušší pochopení činnosti programu. Pro popis diskrétní simulace používáme konečné automaty a regulární gramatiky. Zmíněné části zde více popisujeme, abychom jednodušeji pochopili popis diskrétní simulace. Systém dále umožnuje úpravu vstupních souborů pro jednodušší práci programu.

Klíčová slova: diskrétní simulace, konečný automat, regulární gramatika

Title: Universal discrete simulator

Author: Jiří Vytasil

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Martin Pergel, Ph.D., Department of Software and Computer Science Education

Abstract: In this thesis we analyzed the design and implementation of the program allowing you to perform discrete-event simulation. Further in the program we implemented visualization of discrete-event simulation. Emphasis is placed mainly on universality. In addition to the program and its description includes information of discrete-event simulation, which allows easier understanding of program operation. For a description of discrete-event simulation using finite-state automata and regular grammars. These parts are described more in order to more easily understand the description of the discrete-event simulation. The system also allows adjustment of input files for easier operation of the program.

Keywords: diskrete event simulation, finite-state automaton, regular grammar

Obsah

Úvod

Cílem bakalářské práce je vytvoření programu umožňujícího provádět diskrétní simulace různých jevů. Předmětem bakalářské práce je také jazyk, který popisuje jednotlivé účastníky diskrétní simulace, a vhodná vizualizace diskrétní simulace.

Zvolili jsme si toto téma, protože jsme chtěli vytvořit srozumitelný program pro diskrétní simulace.

V programu je kladen důraz především na univerzalitu. Univerzalitu jsme zaručili při použití konečného automatu a regulární gramatiky jako základu jazyka. Více o univerzalitě v 3.2.1.

Využitím konečného automatu a regulární gramatiky jsme použili již známé prostředky, takže je popis diskrétní simulace jednodušší na pochopení.

Vytvořili jsme program, který se dá využít pro vizualizaci diskrétní simulace. Dá se ale také využít pouze její část reprezentující práci diskrétní simulace, která se dá zavolat i z jiné aplikace.

1. Simulace

Simulace je proces návrhu modelu reálného systému a provádění pokusů s tímto modelem za účelem porozumění chování systému nebo vyhodnocení různých strategií (v mezích daných kritériem nebo souborem kritérií) pro provoz systému. [2]

R. E. Shannon

V definici je zmíněno několik pojmů, které si zde objasníme. Mezi tyto pojmy patří systém a model. Jelikož k provádění simulace je potřeba simulátor, tak si tento pojem také dále popíšeme.

Pod pojmem **systém** si představme skupinu prvků, které se nacházejí ve vzájemné interakci. Systém, který se nemění s časem je neměnný (statický), kdežto ten, který se mění, je dynamický. [2]

Modelem rozumíme reprezentaci nějakého objektu. Model splňuje základní vztahy, vlastnosti a zákony systému, který je jeho vzorem.

Rozlišujeme několik typů modelů. Fyzický a matematický. Fyzický model reprezentuje například repliku, prototyp továrny a další. Matematický model reprezentuje například analytické modely čekání ve frontě, lineární programy, simulace a další.

Model lze užít například ke studování chování systému ve fázi návrhu, předtím než jsou takové systémy vytvořeny. Nebo jej lze užít ke komunikaci návrhu systému. Anebo k předpovědi výkonu nových systémů za lišících se okolností. Případně můžeme model použít na otázky „co kdyby“ z reálného světového systému. [2]

Simulátor reprezentuje počítačový program, který napodobuje jak vnitřní chování reálného světového systému, tak i vstupní procesy, které řídí nebo kontrolují simulovaný systém.

Simulátor výstupu je soubor měření starající se o pozorovatelné reakce a výkon systému. Měření jsou jediné odhady toho, co reálná světová měření vlastně jsou/budou.

Teď si řekneme, proč je dobré simulovat. Ve většině případů raději simulujeme než experimentujeme s reálným světovým systémem a to hned z několika důvodů. Jedním z důvodů je to, že systém dosud neexistuje. Dalším důvodem je, že experimentování se!systémem je nákladné, časově náročné, příliš nebezpečné. Nakonec někdy může být experimentování se systémem nevhodné, například při plánování katastrofy. [2]

V této práci pracujeme převážně s diskrétní simulací. Ale mimo diskrétní simulace existují i další druhy simulací, například spojitá simulace. Tyto dvě simulace se liší především v pohledu na časovou osu.

1.1 Spojitá simulace

Spojitá simulace je založena na změnách rovnoměrně vzdálených v čase. Nejsou zde privilegované okamžiky.

Na simulaci nahlížíme v pravidelných intervalech, ať už se děje něco zajímevého, či nikoliv.

1.2 Diskrétní simulace

Diskrétní simulace představuje modelování, simulování a analýzu systémů s využitím výpočetních a matematických technik zatímco vytváří model koncepčního rámce, který popisuje systém. Systém je simulován provedením experimentu pomocí počítačového uskutečnění modelu a je analyzován, aby vyvodil závěry z výstupu, který pomáhá při procesu rozhodování.

Diskrétní simulace kvantitativně představuje skutečný svět, simuluje jeho dynamiku na principu událostí a generuje detailní zprávu o výkonu. Kvůli dostupnosti výkonného počítače se již dávno stala jedním z běžných počítačových rozhodovacích nástrojů. [1]

Definice. *[7] Diskrétní simulace je definována třemi atributy:*

- stochasticita – *aspoň některé proměnné systému jsou náhodné,*
- dynamičnost – *časový vývoj proměnných systému je důležitý,*
- události – *významné změny v proměnných systému jsou spojené s událostmi, které nastanou pouze v samostatných (jednotlivých) časových případech.*

Komponenty diskrétní simulace

Diskrétní simulace využívá některé komponenty, které reprezentují diskrétní systém. Mezi tyto komponenty patří: čas, generátor náhodných čísel, statistiky, konečné podmínky a kalendář událostí.

Představme si časovou přímku pro *čas*, který ubíhá uvnitř simulace. Na této přímce máme spoustu bodů odpovídajících událostem. Uvnitř simulace potřebujeme tyto události projít a zpracovat v pořadí, jak leží na přímce. Před spuštěním simulace jsme naplánovali několik událostí. Další vznikly jako důsledek předešlých událostí.

Simulace potřebuje generovat pseudonáhodné veličiny, které závisí na modelu. Toto je splněno jedním, nebo více *pseudonáhodnými generátory*. Pseudonáhodná čísla se využívají k napodobení reálných podmínek. Program umožnuje generovat předmětu a procesy.

Výstupem simulace jsou *statistická data* získaná při simulaci, která musíme dále zpracovat, abychom získali výsledné informace. Program negeneruje statistiky.

Pro simulaci je důležité, aby mohla někdy skončit. Proto je nutné zavést *koncové podmínky*, kdy simulace skončí. Např. v čase t nebo po provedení n událostí.

Simulace udržuje alespoň jeden *kalendář událostí*. To je někdy nazváno nevyřízený soubor událostí. Sděluje události, které nejsou vyřízeny v důsledku dříve simulované události, ale ještě musí být sami simulovány. Událost je popsána časem, ve kterém nastane, a typem, který indikuje kód, který bude použit k simulaci té události. Je běžné, že kód události má své parametry, v tom případě obsahuje popis události také parametry ke kódu události. Když jsou události okamžité, jsou činnosti, které se v průběhu času rozšiřují, modelovány jako sekvence událostí. Nějaké struktury simulace dovolují času událostí být stanoven jako interval, který dává čas začátku a čas konce každé události. [7]

Ukážeme si diskrétní simulaci na příkladu. Příkladem bude výtah mezi třemi patry.

Příklad 1. *Máme tři patra a výtah, který mezi těmito patry jezdí.*

Pomocí pseudonáhodného generátoru si vygenerujeme lidi, kteří budou přicházet do náhodných pater v náhodném čase a pojedou do náhodného patra.

Nyní se při zavolání výtahu vytvoří v kalendáři událost s informací, v jakém čase bude výtah v tom patře. Následně se z kalendáře vybere událost s nejnižším časem. Při výběru události z kalendáře nastavíme čas simulace na čas události. Tím jsme se posunuli na časové přímce. Opět se v tomto čase něco provede, například nástup lidí do výtahu a naplánuje se další událost do kalendáře.

Toto se opakuje, dokud neukončíme simulaci, například již nejsou žádní lidé, kteří by čekali na výtah, nebo čas překročil danou mez.

2. Konečný automat a gramatika

2.1 Konečný automat

Definice

- „Abeceda je libovolná konečná neprázdná množina prvků označovaná písmenem Σ. Prvky abecedy se nazývají symboly nebo též znaky abecedy.“ [14]
- „ Slovo v je libovolná konečná posloupnost symbolů dané abecedy Σ. Posloupnost $a_1, a_2, ..., a_n$, kde a_i (pro $i = 1, 2, ..., n$) $\in \Sigma$, je možno zapsat zkráceně $a_1a_2...a_n$, pokud tím nemůže nastat nedorozumění.“ [14]
- „Σ^* = množina všech slov v abecedě Σ.“[11]
- „Σ^+ = množina všech neprázdných slov v abecedě Σ.“[11]
- „$\Sigma^* = \Sigma^+ \cup \{\lambda\}$“[11]
- „Délka slova v je počet znaků, které slovo v tvoří.“[14]
- „Prázdné slovo neobsahuje žádný symbol a značí se znakem λ.“[14]
- „Prefix slova se označuje jeho začátek neboli předpona, sufix slova poté obdobně značí konec nebo-li příponu.“[14]
- „Podslovo je pak jakákoliv část slova.“[14]
- „Prázdné slovo λ je současně prefixem, sufixem i podslovem.“[14]
- „Jsou-li u, v, w, x slova nad abecedou Σ a $u = vwx$, pak je slovo w podslovem slova u. Když je $v = \lambda$, pak je w prefixem, kdyz je $x = \lambda$, pak je w sufixem.“ [14]

Formální jazyk

Definice. *„Formální jazyk nad abecedou Σ je libovolná množina slov nad Σ (jazyky nad Σ jsou tedy právě podmnožiny Σ^*). Jazyk se obvykle značí písmenem L (s indexy)..“[12]*

Deterministický a nedeterministický konečný automat

Definice. *[11] „Deterministický konečný automat je definován jako uspořádaná pětice $(Q, \Sigma, \delta, q_0, F)$, kde:*

- *Q je konečná neprázdná množina stavů.*
- *Σ je konečná neprázdná množina vstupních symbolů, tzv. vstupní abeceda.*
- *δ je přechodová funkce, $\delta : Q \times \Sigma \to Q$.*
- *q_0 je počáteční neboli iniciální stav, $q_0 \in Q$.*
- *F je množina koncových neboli přijímacích stavů, $F \subseteq Q$.“*

Definice. *[11] „Nedeterministický konečný automat je definován jako uspořádaná pětice $(Q, \Sigma, \delta, S, F)$, kde:*

- *Q je konečná neprázdná množina stavů.*
- *Σ je konečná neprázdná množina vstupních symbolů, tzv. vstupní abeceda.*
- *δ je přechodová funkce, $\delta : Q \times \Sigma \to P(Q)$.*
- *S je množina počátečních (iniciálních) stavů, $S \subseteq Q$.*
- *F je množina koncových neboli přijímacích stavů, $F \subseteq Q$.“*

Reprezentace konečného automatu

Existuje několik reprezentací konečného automatu. Zde uvedeme 4 způsoby, které jsou popsány v [8, 9, 10, 11, 14].

Výčet prvků

Každý množinový člen dané pětice je určen výčtem prvků, které obsahuje. U přechodové funkce jsou uvedena všechna pravidla, jimiž se řídí.

Příklad: Konečný automat přijímací slova začínající *aab* a končící *aab*.

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{a, b\}, \delta, q_0, \{g_6\})$$

kde přechodová funkceje určena Tabulkou 2.1

$\delta(q_0, a) = q_1$	$\delta(q_0, b) = q_7$	$\delta(q_4, a) = q_5$	$\delta(q_4, b) = q_3$
$\delta(q_1, a) = q_2$	$\delta(q_1, b) = q_7$	$\delta(q_5, a) = q_5$	$\delta(q_5, b) = q_6$
$\delta(q_2, a) = q_7$	$\delta(q_2, b) = q_3$	$\delta(q_6, a) = q_4$	$\delta(q_5, b) = q_3$
$\delta(q_3, a) = q_4$	$\delta(q_3, b) = q_3$	$\delta(q_7, a) = q_7$	$\delta(q_7, b) = q_7$

Tabulka 2.1: Přechodovou funkci reprezentujeme výčtem všech přechodů.

Tabulka

Další reprezentace je pomocí tabulky. Záhlaví sloupců tvoří symboly vstupní abecedy a záhlaví řádků stavy automatu.

Označení stavů:

- Počáteční stavy symbolem $\rightarrow$.
- Přijímací stavy symbolem $\leftarrow$.
- Stavy, které jsou současně počátační i přijímací, symbolem. $\leftrightarrow$.

Příklad: (Totožný s příkladem u výčtu prvků.)

		a	**b**
$\rightarrow$	q_0	q_1	q_7
	q_1	q_2	q_7
	q_2	q_7	q_3
	q_3	q_4	q_3
	q_4	q_5	q_3
	q_5	q_5	q_6
$\leftarrow$	q_6	q_4	q_3
	q_7	q_7	q_7

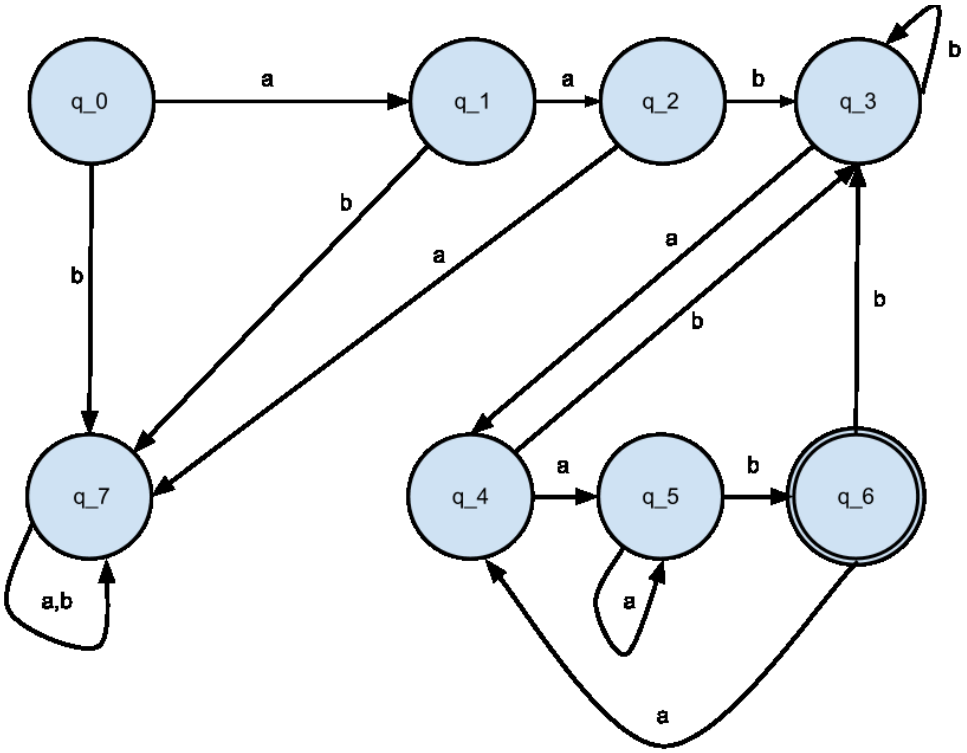
Tabulka 2.2: Konečný automat reprezentovaný tabulkou. V prvním sloupci šipky určují stavy počáteční nebo přijímací. Ve druhém jsou názvy stavů. A ve třetím jsou stavy, kam se dostaneme přes slovo z abecedy zmíněné v záhlaví.

Stavový diagram

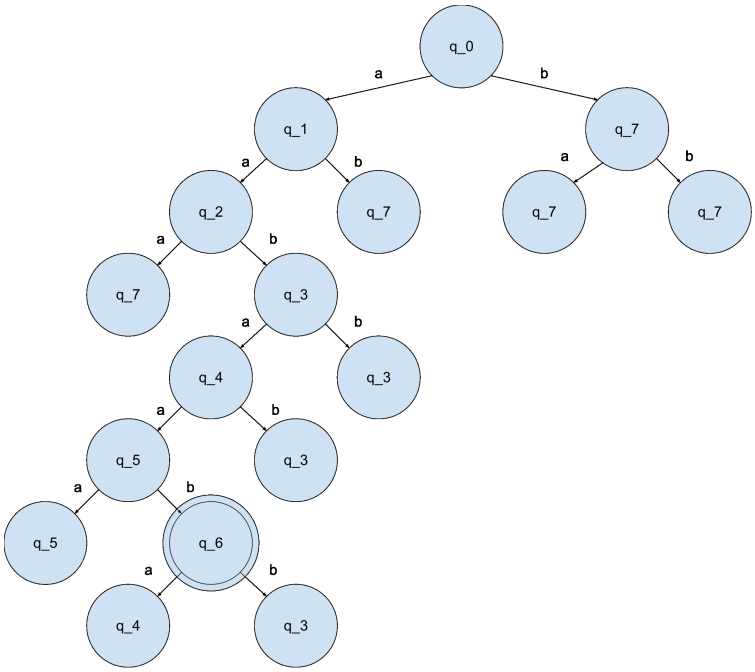
Stavový diagram je tvořen množinou vrcholů představujících stavy automatu a orientovanými hranami mezi těmito vrcholy. Hrany jsou ohodnoceny symboly ze vstupní abecedy. Pokud vede ze stavu q_i do q_j hrana se symbolem a, pak konečný automat ze stavu q_i po přečtení symbolu a přejde do stavu q_j.

Počáteční stavy jsou označeny $\rightarrow$ směřující do stavu.
Přijímací buď dvojitým kruhem, nebo $\rightarrow$ směřující ze stavu.

Příklad: Obrázek 2.1.



Obrázek 2.1: Stavový diagram



Obrázek 2.2: Stavový strom

Stavový strom

Stavový strom je tvořen množinou vrcholů představujících stavy automatu a hranami mezi těmito vrcholy. Hrany jsou ohodnoceny symboly ze vstupní abecedy. Z každého vrcholu, který není listem, vychází právě tolik hran, kolik má příslušný stav následníků. Pokud nějaký stav odpovídá více vrcholům, pak hrany vycházjí jen z jednoho z nich.

Počátečním stavem je kořen stavového stromu. Přijímací stavy jsou označeny buď dvojitým kruhem, nebo $\rightarrow$ vedoucí z vrcholu.

Příklad: Obrázek 2.2.

2.2 Gramatika

Regulární gramatika

Definice. *[14] „Formální gramatikou nazýváme čtveřici (V_N, V_T, S, P):*

- *V_N je neprázdná konečná množina neterminálních symbolů,*
- *V_T je konečná množina terminálních symbolů,*
- *$S \in V_N$ je počáteční neterminální symbol,*
- *P je konečná množina přepisovacích pravidel.“*

„Množiny V_N a V_T jsou disjunktní, tj. $V_N \cap V_T = \emptyset$. V je tzv. celková abeceda, tj. $V = V_N \cup V_T$.“[14]

„Regulární gramatika dovoluje pouze pravidla ve tvaru $X \rightarrow wY,\ X \rightarrow w$, kde $X, Y \in V_N,\ w \in V_T^*$.“[11]

Přepisovací systémy

„Přepisovacím systémem nazýváme dvojici $R = (V, P)$, kde

- V je konečná abeceda,
- P je konečná množina přepisovacích pravidel.“[11]

„Přepisovací pravidlo je uspořádaná dvojice slov (α, β), kde α je slovo, které obsahuje alespoň jeden neterminální symbol. Pravidlo (α, β) se obvykle zapisuje ve tvaru $\alpha \rightarrow \beta$. Pravidla se stejnou levou stranou lze zapsat i takto:“[14]

$$\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, ..., \alpha \rightarrow \beta_n \ lze\ zapsat\ jako\ \alpha \rightarrow \beta_1|\beta_2|...|\beta_n$$

„Říkáme, že w se přímo přepíše na z (píšeme $w \Rightarrow z$), jestliže:

$$\exists u, v, x, y \in V^* \quad w = xuy, z = xvy \text{ a } (u \rightarrow v) \in P.$$

Říkáme, že w se přepíše na z (píšeme $w \Rightarrow^* z$), jestliže:

$$\exists u_1, u_2, ..., u_n \in V^* \quad w = u_1 \Rightarrow u_2 \Rightarrow ... \Rightarrow u_n = z.$$

Posloupnost $u_1, ..., u_n$ nazýváme odvozením.“[11]

Jazyk

Definice. *[14] „Jazyk generovaný gramatikou G je jazyk* $L(G) = \{v | v \in T^* \wedge S \Rightarrow^* v\}$*. Slovo v tedy patří do L(G), pokud platí:*

- *slovo v se skládá pouze z terminálních symbolů,*
- *slovo v lze odvodit z počátečního symbolu S.“*

2.3 Převod regulární gramatiky na konečný automat

Zde si uvedeme dvě věty ukazující převod z regulární gramatiky na konečný automat. Uvádíme je proto, že jsou předmětem implementace. Věty i s důkazy jsou převzaty z [11].

Věta 1. *[11] „Ke každé regulární gramatice* $G = (V_N, V_T, S, P)$ *existuje ekvivalentí gramatika G', která obsahuje pouze pravidla ve tvaru:* $X \to aY$ *a* $X \to \lambda$*.“*

Důkaz. definujme $G' = (V'_N, V'_T, S, P')$, kde pravidla P' získáme takto:

P	P'
$X \to aY$	$X \to aY$
$X \to \lambda$	$X \to \lambda$
$X \to a_1...a_nY$	$X \to a_1Y_2, Y_2 \to a_2Y_3, ..., Y_n \to a_nY$
$Z \to a_1...a_n$	$Z \to a_1Z_1, Z_1 \to a_2Z_2, ..., Z_n \to \lambda$

($Y_2, ..., Y_n, Z_1, ..., Z_n$ jsou nové neterminály - pro každé pravidlo jiná sada)

zbývá $X \to Y$
definujme $U(X) = \{Y | Y \in V_N \wedge X \to^* Y\}$
efektivní postup $U_1 = \{Y | (X \to Y) \in P\}, U_{i+1} = U_i \cup \{Y | (Z \to Y) \in P, Z \in U_i\}$

$X \to Y$	$X \to w$ pro všechna $Z \to w$ z P' a $Z \in U(X)$

□

Věta 2. *[11] „Ke každé regulární gramatice* $G = (V_N, V_T, S, P)$ *existuje konečný automat* $A = (Q, \Sigma, \delta, q_0, F)$ *takový, že* $L(A) = L(G)$*.“*

Důkaz. L(G) je nějaká regulární gramatika obsahující pouze pravidla ve tvaru:

$$X \to aY \quad a \quad X \to \lambda$$

Definujme nedeterministický konečný automat $A = (V_N, V_T, \delta, \{S\}, F)$, kde:

- $F = \{X | (X \to \lambda) \in P\}$
- $\delta(X, a) = \{Y | (X \to aY) \in P\}$

Ještě $L(G) = L(A)$?

1) $\lambda \in L(G) \Leftrightarrow (S \to \lambda) \in P \Leftrightarrow S \in F \Leftrightarrow \lambda \in L(A)$

2) $a_1...a_n \in L(G)$
$\Leftrightarrow$ existuje odvození $(S \Rightarrow a_1X_1 \Rightarrow ... \Rightarrow a_1...a_nX_n \Rightarrow a_1...a_n)$
$\Leftrightarrow \exists X_0, ..., X_n \in V_N$ tž. $\delta(X_i, a_{i+1}) \ni X_{i+1},\ X_0 = S, X_n \in F$
$\Leftrightarrow\ a_1...a_n\ \in\ L(A)$

□

3. Analýza projektu

3.1 Programovací jazyk

Jelikož je cílem vytvořit jazyk pro diskrétní simulace, nabízí se objektově orientované jazyky jako Java, C++, C#. Dále je cílem kromě jazyka i vizualizace simulace. U C++ je designování těžkopádnější a to i včetně knihovny QT. Z tohoto důvodu se C++ příliš nehodí.

Nakonec jsme si jako jazyk pro implementaci zvolili C#.

3.2 Rozhodovací konflikty

V této sekci se budeme zabývat hlavními rozhodovacími konflikty, které bylo potřeba rozřešit.

3.2.1 Základ jazyka

Velmi důležitou otázkou tohoto programu je univerzálnost. Otázkou tedy je, co použít jako jazyk, aby byla univerzálnost zachována.

Odpovědí je několik, ale dají se zúžit do dvou hlavních kategorií. První kategorií je vymyslet si vlastní jazyk, který na nic nenavazuje. Do druhé kategorie bych zařadil jazyky, u kterých jako základ využijeme již známé prostředky.

U první kategorie mezi hlavní důvody můžeme zařadit jednodušší programování, jelikož si můžeme vytvořit jazyk, který bude vyhovovat nám jako programátorům.

Oproti tomu v druhé kategorii, pokud si vybereme dobře, nám odpadne řešení univerzálnosti. Mezi další výhody se dá zařadit jednodušší rozšiřitelnost, protože si jako základ vybereme již známé prostředky, takže někteří uživatelé budou moci program používat s minimem obtíží.

Z výše zmíněných důvodů jsme zvolili druhou kategorii.

Otázkou je, co použít jako základ jazyka.

Jako základ jsme uvažovali o automatu, gramatice, nebo využití některého z programovacích jazyků (např. C#, Java, ...). Využití programovacího jazyku jsme si nevybrali, protože bychom jen vytvořili jiný interpret některého programovacího jazyka. Jelikož automat a gramatika jsou vzájemně na sebe převoditelné, tak jsme se rozhodli pro využití obou těchto možností. Jako hlavní pro simulaci jsme se rozhodli využít automat. Gramatiku nepoužíváme pro simulaci, ale jen ji převedeme na automat.

3.2.2 Automat

Automatů existuje několik druhů. Konečné automaty, zásobníkové automaty, lineárně omezené automaty a Turingovy stroje. [11] Otázkou je, který použijeme.

Doporučuje se, pokud je to možné, použití jednoduššího automatu. Nejjednodušším automatem je konečný automat. Podíváme se, zda nám konečný automat bude stačit.

Ukážeme si na několika příkladech, zda je dokážeme vyjádřit pomocí konečného automatu. Nebudeme řešit, zda příklady vyjádříme pomocí nedeterministického, nebo deterministického konečného automatu, protože jsou na sebe navzájem převoditelné. Příklady ukážeme jen pomocí automatu a nikoli pomocí gramatiky, protože jsou na sebe navzájem převoditelné. Více v [11].

Příklad 2. *Konečný automat reprezentující pohyb výtahů. Pro zjednodušení budeme počítat se 4 stavy.*

$$M = (Q = \{patro_0, patro_1, patro_2, patro_3\}, \{n, d, o\}, \delta, patro_0, Q)$$

kde n = jet nahoru, d = jet dolů, o = otevřít dveře výtahu.
Přechodová funkce δ vypadá následovně:

$\delta(patro_0, n) = patro_1$	$\delta(patro_0, o)) = patro_0$	
$\delta(patro_1, n) = patro_2$	$\delta(patro_1, d) = patro_0$	$\delta(patro_1, o) = patro_1$
$\delta(patro_2, n) = patro_3$	$\delta(patro_2, d) = patro_1$	$\delta(patro_2, o) = patro_2$
$\delta(patro_3, d) = patro_2$	$\delta(patro_3, o) = patro_3$	

Obdobně lze simulovat autobusovou dobravu, jen místo pater použijeme zastávky.

Příklad 3. *Konečný automat reprezentující převoz nákladu. Opět pro zjednodušení budeme počítat s méně stavy.*

$$M = (\{s_0, s_1, s_2, s_3\}, \{n, v, j\}, \delta, s_0, s_3)$$

kde n = naložit náklad, v = vyložit náklad, j = jet

s_0 = *prázdný nákladní vůz v místě A* s_1 = *naložený vůz v místě A*
s_2 = *naložený vůz v místě B* s_3 = *vyložený vůz v místě B*

Přechodová funkce δ vypadá následovně:

$\delta(s_0, n) = s_1$	$\delta(s_1, j) = s_2$
$\delta(s_2, v) = s_3$	$\delta(s_3, j) = s_0$

Příklad 4. *Konečný automat reprezentující křižovatku.*

$$M = (\{s_0, s_1, s_2, s_3\}, \{r, l, t\}, \delta, \{s_0, s_1, s_2, s_3\}, \{s_0, s_1, s_2, s_3\})$$

kde r = jet doprava, l = jet doleva, t = jet rovně.

s_0 = *severní vjezd/výjezd do/z křižovatky*
s_1 = *západní vjezd/výjezd do/z křižovatky*
s_2 = *jižní vjezd/výjezd do/z křižovatky*
s_3 = *východní vjezd/výjezd do/z křižovatky*

Přechodová funkce δ vypadá následovně:

$\delta(s_0, r) = s_1$	$\delta(s_0, l)) = s_3$	$\delta(s_0, t) = s_2$
$\delta(s_1, r) = s_2$	$\delta(s_1, l) = s_2$	$\delta(s_1, t) = s_3$
$\delta(s_2, r) = s_3$	$\delta(s_2, l) = s_1$	$\delta(s_2, t) = s_0$
$\delta(s_3, r) = s_0$	$\delta(s_3, l) = s_0$	$\delta(s_3, t) = s_1$

Typicky potřebujeme reprezentovat data, na kterých dotyčná simulace běží (například lidi jedoucí výtahem, nebo náklad převážený kamiony). Tento problém jsme vyřešili přidáním tzv. předmětů.

3.2.3 Nedeterminismus automatu

Konečný automat může být deterministický, ale i nedeterministický. Otázkou tedy je, jak se s nedeterminismem vypořádat.

Jedním řešením by mohlo být nedeterminismus zakázat, ale to bychom nemohli používat tento program pro většinu simulací.

Jako další řešení bychom mohli použít BFS, pomocí kterého bychom v simulaci řešili všechny možné cesty. V tomto řešení bychom dokonce dostali nejlepší možný čas. Ale jako nevýhodu bychom mohli označit menší kontrolovatelnost a při vizualizaci nutnost nejdříve simulaci dokončit. Dokončení je při vizualizaci nutné z toho důvodu, abychom mohli správně uživateli signalizovat přechod z jednoho stavu do jiného.

Nakonec jsme se rozhodli pro následující řešení nedeterminismu. Při nedeterminismu se zavolá podmínka, kterou uživatel definoval. Zde přechody můžeme mnohem více kontrolovat a nemusíme simulaci před vizualizací dokončit, takže při složitější simulaci je menší pravděpodobnost čekání.

V podmínkách jsme povolili operátory *and* a *or*, které jsou vyhodnocovány zleva doprava a byla jim dána stejná priorita. Dále kromě těchto operátorů můžeme použít závorky, které mají vyšší prioritu než operátory *and* a *or*. Nakonec jsme se rozhodli přidat do podmínek i proměnnou `last`, která reprezentuje poslední použité slovo z abecedy. Proměnná zjednodušuje psaní podmínek, například u výtahu - pokud otevřeme dveře pro nastoupení/vystoupení lidí, tak následně výtah přejede do jiného patra.

3.2.4 Více přechodových funkcí

Jelikož zde nemáme předem danou sekvenci znaků z abecedy, u které chceme zjistit, zda je přijímána naším automatem, tak musíme řešit i problém více přechodových funkcí vedoucí z jednoho stavu.

Tohoto problému se zbavíme stejně jako problému s nedeterminismem. Opět použijeme podmínky, kde to můžeme lépe specifikovat.

3.2.5 Gramatika

Jako druhou část zadávání diskrétní simulace jsme si zvolili gramatiku. Těch je také několik druhů: regulární, bezkontextová, kontextová a typu 0. [11] Otázkou tedy je, jakou si vybereme.

Jelikož v našem programu budeme gramatiku převádět na konečný automat, logicky se nabízí regulární gramatika. U regulární gramatiky použijeme pouze pravidla typu:

- $X \rightarrow aY$

  a terminál

  X, Y neterminál

- $X \rightarrow \lambda$

  zde v programu λ nahrazena \$

  X neterminál

3.2.6 Vizualizace

Součástí programu je i vizualizace diskrétní simulace. Otázkou tedy je, jak vizualizovat informace z diskrétní simulace.

Jako jedna z variant se nabízela možnost zobrazení stavů s informacemi o předmětech, které se zde vyskytují. Dále zobrazit procesy s bližšími informacemi, které přechází z jednoho stavu do jiného. Toto zobrazení je sice nejjednodušší na čtení průběhu diskrétní simulace, ale při zobrazení na menších obrazovkách by mohl nastat problém při zobrazení většího množství stavů, ve kterých bychom zobrazovali ještě procesy. Proto jsme se pro toto zobrazení nerozhodli.

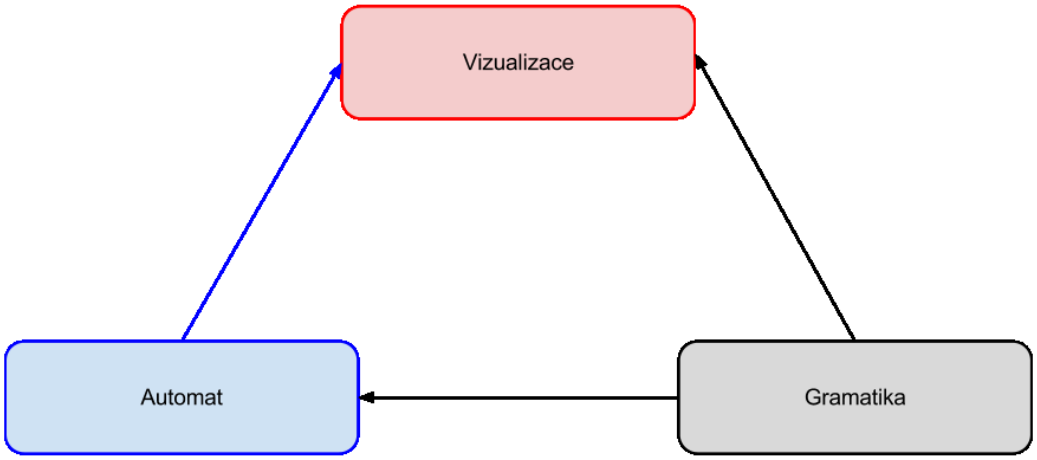
Další variantou je zobrazit stavy a procesy zvlášť. Tímto způsobem snížíme velikost stavů. Ve stavech zobrazíme informace o předmětech, které se zde vyskytují, a název procesu, který se zde v tento čas vyskytuje. Procesy zobrazíme ve vedlejším okně jako seznam s bližími informacemi. Dále zobrazíme informace o kalendáři událostí, abychom měli lepší představu o nadcházejících událostech. Pro tento způsob zobrazení jsme se rozhodli z důvodů, že je celkem jednoduché z něj vyčíst průběh diskrétní simulace a ani zobrazení většího množství stavů není velký problém.

4. Implementace

4.1 Popis jednotlivých součástí

Celkový program obsahuje několik součástí pokrývajících jednotlivé části. Jsou jimi:

- Automat
- Gramatika
- Vizualizace



Obrázek 4.1: Diagram zachycuje vztahy mezi částmi programu. Pokud vede šipka od části X do části Y, tak existuje v Y reference na X.

Projekt Automat je knihovna reprezentující konečný automat, jak jeho práci, tak i načtení. Více se dozvíme v kapitole 4.2.

Projekt Gramatika je knihovna reprezentující regulární gramatiku. Tento projekt načte gramatiku a připraví ji na převod na konečný automat. Více se dozvíme v kapitole 4.3.

Projekt vizualizace se stará o zobrazení diskrétní simulace. Více se dozvíme v kapitole 4.4.

4.2 Konečný automat

Zde budeme popisovat část programu popisující konečný automat.

Hlavní třídy

Zde si uvedeme některé hlavní třídy a co reprezentují v rámci konečného automatu:

Mezi hlavní třídy patří třídy reprezentující jednotlivé části konečného automatu. Jednou z nich je třída `Stav`, která reprezentuje jednotlivé stavy konečného automatu. Dále tu je třída `Abeceda`, která reprezentuje jednotlivá slova abecedy. Třída `PrechodovaFunkce` reprezentuje přechodové funkce konečného automatu. Dále může obsahovat název podmínky, která je potřebná pro správné rozhodnutí.

Třída `Predmet` reprezentuje předměty automatu, které mají procesy přemístit z jednoho stavu do jiného. Třída `Proces` reprezentuje jednotlivé procesy konečného automatu. Tyto procesy představují konatele činností. Třída `Podminka` reprezentuje podmínky, které jsou potřebné pro správné zvolení přechodové funkce.

Třída `Automat` obsahuje datové struktury pro ukládání zmíněných tříd. Dále provádí přechod z jednoho stavu do jiného.

Práce automatu

Třída `Automat` reprezentuje konečný automat a provádí jednotlivé kroky diskrétní simulace. Obsahuje datové struktury reprezentující např. abecedu, přechodové funkce, atd. Provádí přechody z jednoho stavu do jiného a při volání podmínky provádí její vyhodnocení. Dále převádí gramatiku na konečný automat. Tato třída je rozdělena do čtyř souborů.

Jeden soubor se zabývá převedením regulární gramatiky na konečný automat. Převod provádí z předpřipravené regulární gramatiky. K převodu používá metody, které z jednotlivých částí regulární gramatiky vytvoří postupně jednotlivé části konečného automatu (např. neterminály převede na stavy). Název souboru je `AutomatGramatika`.

Další soubor se zabývá vyhodnocením podmínky. Tento soubor je určen pro vyhodnocení volané podmínky a určení správné přechodové funkce. K tomu je také určen seznam již volaných podmínek, abychom se nezacyklili. Při vyhodnocování podmínek vytváříme převodní tabulku, která slouží pro převod pseudoparametrů na parametry reálné. Název souboru je `AutomatVyhodnoceniPpodminky`.

V dalším souboru řešíme přechod z jednoho stavu do jiného. K tomu využíváme také soubor určený k vyhodnocování podmínek.

Zde zjistíme aktuální událost a z této události dostaneme informace o procesu, stavu. Z těchto informací určíme přechodovou funkci a slovo z abecedy. Pokud se nám nepodaří jednoznačně určit přechodovou funkci, zjistíme si z ní název podmínky, která ji určí, a necháme ji vyřešit v souboru, který se zabývá vyhodnocováním podmínek. U slova z abecedy se podíváme, zda nakládá předměty na procesy nebo vykládá předměty z procesů. Pokud ano, zavolají se metody pro nakládání či vykládání.

Dále pokud máme definován výstup do souboru, provedeme v tomto souboru i tuto činnost. Název souboru je `AutomatZpracovani`.

Poslední soubor obsahuje datové struktury, které reprezentují všechny důležité části automatu, jako např. předměty, procesy, slova abecedy, přechodové funkce, podmínky, stavy, chyby. V tomto souboru jsou také metody viditelné zvenčí, které slouží např. pro vytvoření instance této třídy, či pro provádění diskrétní simulace. Zde si následně vytváříme iniciální události, kde se jednotlivé procesu budou nacházet. A provádíme si tu kontrolu, aby konečný automat obsahoval všechny důležité části. Název souboru je `AutomatInicializace`.

Slova z abecedy

Třída `Abeceda` je určena pro reprezentaci slov z abecedy. Slovo určuje činnost simulace.

Zde si pamatujeme název slova z abecedy, resp. činnosti simulace, číslo řádku využívané při nalezení chyby a hodnoty indikující, zda je možné u této činnosti nakládání předmětů, vykládání předmětů, nebo si tuto činnost budeme pamatovat pro vyhodnocování podmínek.

Kalendář událostí

Třída `Kalendar` reprezentuje jednotlivé události. Zde si pamatujeme nejbližší události pro každý proces. Jednotlivé události jsou instancí třídy `Událost`. Tato třída slouží pro vytváření nových událostí a odstraňování již proběhlých událostí.

Podmínky pro správné vyhodnocování

Třída `Podminka` je určená pro reprezentaci podmínek. Podmínky slouží pro vybrání správné přechodové funkce.

Zde si pamatujeme název podmínky, seznam pseudoparametrů, za které při volání dosadíme reálné. Dále je tu výraz pro vyhodnocení, který vrací `true`, nebo `false`. Nakonec tu jsou větve `then` a `else` a čísla řádků, na kterých podmínka začíná a končí v souboru. Pro větev `then` a `else` používáme abstraktní třídu, kam ukládáme buď větev reprezentující přechodovou funkci, nebo větev reprezentující volání další podmínky.

Předměty

Třída `Predmet` je určena pro reprezentaci předmětů. Tyto předměty jsou následně procesy převáženy z jednoho stavu do jiného.

Zde si pamatujeme název předmětu, množství, čas, kdy s předmětem můžeme začít pracovat, startovní stav, stav, kam je potřeba dopravit, seznam stavů reprezentující přestupy a číslo řádku při případném výskytu chyby.

Procesy

Třída `Proces` je určena pro reprezentaci procesů. Tyto procesy přecházejí z jednoho stavu do jiného.

Zde si pamatujeme název procesu, kapacitu, startovní stav, čas, kdy proces začne pracovat, seznam časů pro abecedu, proměnnou pro uložení naposledy použité abecedy a číslo řádku při případném výskytu chyby.

Přechodové funkce konečného automatu

Třída `PrechodovaFunkce` je určená pro reprezentaci přechodové funkce konečného automatu.

Zde si pamatujeme název stavu, ze kterého je přechodová funkce volána, zda je v přechodové funkci volána nějaká podmínka pro zjištění správné přechodové funkce, číslo řádku ze souboru pro případné vypsání chyby.
Následně si zde pamatujeme buď název slova z abecedy a stav, kam se přesuneme, nebo název podmínky, kterou budeme volat, a seznam předávaných parametrů.

Z předchozího plyne, že nikdy nejsou vyplněny všechny proměnné. Pokud přechodová funkce nevolá žádnou podmínku, tak není vyplněn seznam parametrů. A pokud podmínku voláme, tak není vyplněn stav, kam se přesuneme.

Stavy automatu

Třída `Stav` je určená pro reprezentaci stavu konečného automatu.

Zde si pamatujeme název stavu a zda je stav přijímací, či nikoliv a dále pro správné označení chyby číslo řádku ze souboru.

Další třídy

Zde popíšeme další třídy, které automat obsahuje. Tyto třídy jsou zmíněné zde, protože se používají jen jednou, nebo jsou jednoduché na vysvětlení.

`CteniSouboru`
Třída přečte soubor a vytvoří položky datových struktur automatu.

Třída obsahuje jednu metodu pro čtení souboru a vytváření instancí jiných tříd. Tyto instance tvoří položky datových struktur automatu.

`Prevoz`

Třída reprezentující jednotlivé položky, které jsou naloženy na procesech.

Proměnné reprezentují název předmětu, cílový stav a počet nákladu na procesu. `Udalost`

Třída reprezentující jednotlivé události.

Proměnné reprezentují čas události, název procesu, slovo abecedy, stav, ve kterém se proces bude nacházet, a náklad. Náklad je reprezentován seznamem instancí třídy `Prevoz`.

`Avetev`

Abstraktní třída sloužící jako předek pro třídy **VetevKonecna** a **VetevPodminka**.

Obsahuje proměnnou reprezentující buď slovo abecedy (*VetevKonecna*), nebo název další podmínky (*VetevPodminka*). Dále obsahuje hlavičku metody **Vyhodnot**.

`VetevKonecna`

Třída reprezentující větev podmínky, která obsahuje přechodovou funkci. Předkem je abstraktní třída *Avetev*.

Metoda obsahuje proměnné pro stavy reprezentující stav odkud a stav kam.

`VetevPodminka`

Třída reprezentující větev podmínky, která obsahuje volání další podmínky. Předkem je abstraktní třída *Avetev*.

Metoda obsahuje seznam reprezentující parametry, které se předají volané podmínce.

`Kontrola`

Tato třída provádí závěrečnou kontrolu před spuštěním simulace automatu. Kontrolujeme zde, zda jsou všechny důležité proměnné nastaveny a zda mají správné názvy.

`GeneratorPredmetu`

Třída pro generování předmětů. Tyto předměty mohou mít nějaké hodnoty náhodné.

`GeneratorProcesu`

Třída pro generování procesů. Procesy mohou mít nějaké hodnoty náhodné.

4.3 Regulární gramatika

Tato část programu obsahuje jen dvě třídy, jelikož se bude následně gramatika převádět na automat a všechna práce související s převedením regulární gramatiky na konečný automat je řešena třídou `Automat`.

Práce gramatiky

Třída `Gramatika` reprezentuje načtení gramatiky ze souboru a následnou úpravu pro převod na konečný automat.

Třída je rozdělena do tří souborů.

Jeden soubor reprezentuje čtení gramatiky ze souboru a následné uložení regulární gramatiky do datových struktur. Zde jsou proměnné pro přijímací stavy, které mohou být zadány buď pomocí automatové syntaxe, nebo přes pravidla, ale ne oběma způsoby. Název souboru je `GramatikaCteniSouboru`.

Další soubor reprezentuje úpravu gramatiky pro převod na konečný automat. Zde si vytvoříme seznamy pro reprezentaci neterminálů, terminálů a dalších částí, které se následně budou jednoduššeji převádět na konečný automat. Také si zde vytváříme podmínky, které použijeme při práci konečného automatu. Název souboru je `GramatikaPrevodNaKA`.

Poslední soubor obsahuje definice datových struktur pro uložení gramatiky. Zde jsou datové struktury pro uložení důležitých informací, např. pravidel, procesů, předmětů a další. Zde vytváříme instanci této třídy a následně upravujeme pravidla, aby šla jednodušeji převést na přechodové funkce konečného automatu. Název souboru je `GramatikaInicializace`.

Pravidla gramatiky

Třída `Pravidlo` reprezentuje pravidla gramatiky.

Obsahuje proměnné pro neterminál, ze kterého je pravidlo voláno, dále terminál reprezentující činnost automatu, neterminál, kam se pomocí tohoto pravidla dostaneme, podmínku, pokud je nutná. Dále obsahuje čísla řádků, kde pravidlo začíná a končí, a odkaz na další pravidlo, pokud existuje.

Zde nebudou vyplněné všechny proměnné, stejně jako u přechodových funkcí konečného automatu.

4.4 Vizualizace

Každá z následujících tříd reprezentuje jedno zobrazované okno.

Uvítací obrazovka

Okno `MainWindow` se zobrazí po spuštění programu. Slouží k otevření prázdného editoru, nebo otevře soubor, nebo se ukončí.

Editor

Okno `EditorAutomat` slouží pro úpravu automatu, nebo gramatiky. Toto okno obsahuje editor z programu `AvalonEdit`, který je pod licencí LGPL-3.0. Obsahuje menu pro otevření souboru, uložení souboru, otevření nového souboru a ukončení programu. Dále jsou v menu položky pro spuštění automatu a gramatiky. Následně z tohoto okna lze spustit diskrétní simulaci.

Chyby

Toto okno se zobrazí, pokud jsou v automatu nebo gramatice chyby. Tyto chyby jsou následně v editoru podtržené červeně.

Zobrazení stavů

Okno `AutomatZobrazeni` se zobrazí, pokud nenastane chyba při čtení automatu nebo gramatiky. Zde se zobrazují stavy s bližšími informacemi o předmětech, které se v nich nacházejí. Dále zobrazuje názvy procesů, které se v tu chvíli v nich nacházejí.

Zobrazení kalendáře

Okno `AutomatKalendar` se zobrazí spolu s okny `AutomatZobrazeni` a `AutomatProcesy`. Zobrazí aktuální čas a události obsažené v kalendáři.

Zobrazení procesů

Okno `AutomatProcesy` se zobrazí spolu s okny `AutomatZobrazení` a `AutomatKalendar`. Zobrazí procesy a informace o jejich nákladu, jako název, množství a cílový stav.

5. Uživatelská dokumentace

5.1 Automat

Vstupní data

Vstupní data pro běh programu musí být uložena v souboru s příponou `.txt`, jiné koncovky program neakceptuje.

Vstupní soubor je rozdělen do několika částí. Rozdělení je jen logické, tedy části nemusejí jít v tomto pořadí, navíc ani jedna konrétní část nemusí být v souboru pohromadě. Soubor obsahuje tyto části:

- Proměnné
- Předměty automatu
- Procesy automatu
- Stavy
- Přijímací stavy
- Abeceda
- Přechodové funkce
- Podmínky
- Vlasní výpis

Komentáře v souboru jsou pouze jednořádkové, tj. od posloupnosti znaků `//` do konce řádku.

Proměnné

Tato část je nepovinná pro běh programu. Lze ji využít pouze ke zjednodušení vyhodnocování podmínek, a při vlastním výpisu.

Proměnných je předem daný počet a vyskytují se jen ve výrazu určeném k vyhodnocení v podmínkách. Proměnná se skládá ze dvou položek oddělených tečkou.

První položkou je název logické části, kam proměnná ukazuje (proces, predmet, abeceda).

Druhá položka pak ukazuje na konkrétní název, na který se můžeme odkázat v logické části, kterou máme určenou první položkou.

Seznam možných kombinací, které lze použít v proměnných:

- `proces.name`

  tato proměnná reprezentuje název aktuálního procesu

- `proces.kapacita`

  tato proměnná nám dovoluje přistoupit k volné kapacitě procesu, který právě vykonává nějakou činnost. Například u výtahu zjistíme, že do něj mohou nastoupit ještě 3 osoby.

- `predmet.name`

  tato proměnná reprezentuje názvy předmětů

- `predmet.kapacita`

  zde zjistíme množství předmětu, které ještě zbývá přemístit

- `predmet.start`

  zde zjistíme, kdy bude předmět připraven k přemisťování

- `predmet.in`

  zde zjistíme, kde předmět začíná svou pouť

- `predmet.out`

  zde zjistíme, kam máme předmět přemístit

- `abeceda.last`

  zde dostaneme poslední slovo z abecedy, které je označeno znakem @

- `aktualni.cas`

  zde dostaneme aktuální čas

- `stav.konec`

  tato proměnná reprezentuje přijímací stavy

- `aktualni.stav`

  tato proměnná reprezentuje aktuální stav, ve kterém se proces nachází.

  tato proměnná lze použít pouze ve vlastním výpisu

Předmět automatu

Tato část je povinná pro běh programu. Tato část nám říká, co se bude během běhu programu přemisťovat z jednoho místa jinam. Také nám říká, kdy můžeme program ukončit. Název předmětu může obsahovat pouze malá písmena, velká písmena, číslice, kulaté závorky `()`, čárku `,`, podtržítko `_` a pomlčku `-`.

Zápis předmětu v souboru:
řádek začíná slovem **co**, za kterým následuje tabulátor. Dále následuje v tomto pořadí:

- název,
- počet,
- startovní čas, kdy s tímto předmětem můžeme pracovat,

- místo, na kterém bude tento předmět čekat na zpracování,
- místo, kam předmět přemístíme,
- volitelně v [ ] seznam stavů, oddělených |, reprezentující přestupy.

Ukázka použití. *Příklad reprezentující vytvoření předmětu s názvem písek.*
co pisek 1000 0 (A) (B) [(C)]

Další možnost zápisu:
řádek začíná slovem **co**, za kterým následuje tabulátor. Dále následuje v tomto pořadí:

- název,
- znak {,
- seznam hodnot, které jsou psány ve stylu atribut=číselná hodnota, kde atribut je:

  `min`, které reprezentuje minimální počet generovaných předmětů (implicitní hodnota je `0`),

  `max`, které reprezentuje maximální počet generovaných předmětů (implicitní hodnota je `MaxInt`),

  `timeMin`, které reprezentuje minimální čas, kdy s tímto předmětem budeme moci začít pracovat (implicitní hodnota je `0`),

  `timeMax`, které reprezentuje maximální čas, kdy s tímto předmětem budeme moci začít pracovat (implicitní hodnota je `MaxInt`),

  `kapacitaMin`, které reprezentuje minimální velikost předmětu (implicitní hodnota je `0`),

  `kapacitaMax`, které reprezentuje maximální velikost předmětu (implicitní hodnota je `MaxInt`),.

- Dále je zde povinně buď dvojice `from` a `to`, nebo `road`, kde:

  `from` je seznam stavů oddělených mezerou, ze kterých se náhodně vybere startovní stav,

  `to` je seznam stavů oddělených mezerou, ze kterých se náhodně vybere cílový stav,

  `road` je seznam cest oddělených |, kde každá cesta obsahuje stavy oddělené mezerou. Cesta obsahuje startovní stav, přestupní stavy a nakonec cílový stav. Cesta se vybírá náhodně.

- Nakonec zde můžeme zapsat hodnotu `unique`, které nám vytvoří jednoznačná (v rámci tohoto generování) jména předmětů. Implicitně nevytváří jednoznačná jména předmětů.
- Znak }.

Hodnoty ve složených závorkách { } musí být oddělené středníkem (;).

Ukázka použití. *Příklad reprezentující vytvoření předmětů s názvem písek.*
co pisek { min=1; max=2; timeMin=150; timeMax=500; kapacitaMin=100; kapacitaMax=200; from=[(A)]; to=[(C)] }
co pisek { min=1; max=2; timeMin=150; timeMax=500; kapacitaMin=100; kapacitaMax=200; road=[(A) (C) (B)] }

Proces automatu

Tato část je povinná pro běh programu. Udává nám konatele, který bude procházet automatem. Název procesu může obsahovat pouze malá písmena, velká písmena, číslice, kulaté závorky `()`, čárku `,`, podtržítko `_` a pomlčku `-`.

Zápis procesu v souboru:
řádek začíná znakem **p**, který je následován tabulátorem. Dále následují tyto části:

- název,
- kapacita,
- startovní místo : startovní čas,
- následuje seznam časů pro jednotlivé znaky z abecedy ve stejném pořadí, v jakém jsou dané v abecedě.

Ukázka použití. *Příklad reprezentující vytvoření procesu s názvem auto.*
p auto 10 (A):0 10 10 10

Další možnost zápisu:
řádek začíná znakem **p**, který je následován tabulátorem. Dále následuje v tomto pořadí:

- název,
- znak {,
- seznam hodnot, které jsou psány ve stylu atribut=číselná hodnota, kde atribut je:

  `min`, které reprezentuje minimální počet generovaných procesů (implicitní hodnota je `0`),

  `max`, které reprezentuje maximální počet generovaných procesů (implicitní hodnota je `MaxInt`),

  `timeMin`, které reprezentuje minimální čas, kdy s tímto procesem budeme moci začít pracovat (implicitní hodnota je `0`),

  `timeMax`, které reprezentuje maximální čas, kdy s tímto procesem budeme moci začít pracovat (implicitní hodnota je `MaxInt`),

  `nosnostMin`, které reprezentuje minimální nosnost procesu (implicitní hodnota je `0`),

  `nosnostMax`, které reprezentuje maximální nosnost procesu (implicitní hodnota je `MaxInt`).

- Dále je zde povinně atribut `from`, který obsahuje v [ ] množinu stavů oddělených mezerou, ze které se vybírá startovní stav,
- nakonec je zde povinný atribut `abeceda`, který obsahuje v [ ] množinu časů oddělených mezerou. Tyto časy se přiřadí jednotlivým slovům z abecedy v pořadí, v jakém byly zadány. Jednotlivé časy mohou být tvaru buď číselná hodnota (udává konkrétní čas), nebo interval (hondnota-hodnota), kde oddělovačem je pomlčka.
- Znak }

Ukázka použití. *Příklad reprezentující vytvoření procesu s názvem nakladak.*
p nakladak { min=1; max=2; timeMin=100; timeMax=200; nosnostMin=25; nosnostMax=50; from=[(A)]; abeceda=[10-20 5-10 5] }

Stavy automatu

Tato část je povinná pro běh programu. Udává nám místa, ve kterých sledujeme diskrétní simulaci. Ve stavech sledujeme, co tam jednotlivé procesy vykonávají. Název stavu může obsahovat pouze malá písmena, velká písmena, číslice, kulaté závorky `()`, čárku `,`, podtržítko `_` a pomlčku `-`.

Zápis stavu v souboru:
řádek začíná znakem **Q**, který je následován tabulátorem. Poté následuje jen jedna část, a to název stavu.
Název stavu nesmí být *Q*.

Ukázka použití. *Příklad reprezentující stav automatu.*
Q (A)

Přijímací stavy

Tato část je povinná pro běh programu. Udávají se tu stavy, ve kterých se musí nacházet procesy, aby simulace mohla být ukončena.

Zápis stavu v souboru:
řádek začíná znakem **F**, který je následován tabulátorem. Poté může nastat jedna ze dvou možností:

- napíšeme název stavu, který budeme chtít mít za přijímací
- použijeme znak Q, který nám říká, že za přijímací stavy chceme použít všechny stavy automatu

Ukázka použití. *Příklad reprezentující přijímací stav automatu.*
F (A)

Abeceda

Tato část je povinná pro běh programu. Udává nám činnosti, jejichž pomocí procesy přechází z jednoho stavu do jiného. Název abecedy může obsahovat pouze malá písmena, velká písmena, číslice, kulaté závorky `()`, čárku `,`, podtržítko `_` a pomlčku `-`.

Zápis abecedy v souboru:
řádek začíná znakem **X**, který je následován tabulátorem. Dále obsahuje povinně název a nepovinně může obsahovat některé z následujících znaků oddělené mezerou.

- znak @ indikuje, že takto označená činnost, která byla použita naposled, bude uložena v proměnné abeceda.last
- znak + indikuje nakládání předmětů na procesy
- znak - indikuje vykládání předmětů z procesů

Ukázka použití. *Příklad reprezentující abecedu.*
X nalozit + @)

Přechodové funkce

Tato část je povinná pro běh programu. Udávají nám přechod z jednoho stavu do jiného stavu pomocí slova z abecedy.

Zápis přechodové funkce v souboru:
řádek začíná znakem **d**, který je následován tabulátorem.
Přechodové funkce v programu jsou dvojího typu:

- přímá přechodová funkce
- přechodová funkce s podmínkou

Pokud z jednoho stavu máme více přechodových funkcí, tak použijeme přechodovou funkci s podmínkou, kde po splnění nějakého počtu podmínek dostaneme správnou přechodovou funkci, pomocí které se dostaneme do výsledného stavu.

Přímá přechodová funkce

Zde reprezentujeme přímý přechod z jednoho stavu do jiného. Funkce obsahuje tuto posloupnost:

- stav, kterého se tato funkce týká,
- slovo z abecedy, které udává, jak se proces dostane do dalšího stavu,
- stav, ve kterém se proces bude nacházet.

Ukázka použití. *Příklad reprezentující přímou přechodovou funkci.*
d (vydejce,vylozen) nalozit (vydejce,nalozen)

Přechodová funkce s podmínkou

Přechodová funkce s podmínkou je přechodová funkce, která se použije, pokud by z jednoho stavu vedlo více různých přechodových funkcí. Funkce obsahuje následující posloupnost:

- stav, kterého se tato funkce týká
- slovo check, které udává, že funkce bude volat podmínku
- název podmínky, která se bude vykonávat
- seznam stavů v [ ], oddělených mezerou, které se předají zmíněné podmínce

Ukázka použití. *Příklad reprezentující přechodovou funkci s podmínkou.*

```
d     (A) check vA [(A) (C)]
```

Podmínky

Tato část je nepovinná pro běh programu. Pokud ze stavu vede více přechodových funkcí, tak nám podmínky pomáhají vybrat jednu, kterou použijeme.

Zápis podmínky v souboru:
řádek začíná slovem **check**, následovaný tabulátorem. Dále obsahuje tuto posloupnost:

- název podmínky, přes který ji budeme volat,
- v kulatých závorkách, seznam parametrů oddělených čárkou,
- za seznamem parametrů je pouze znak {, který nám určuje začátek samotné podmínky,
- podmínku, která může obsahovat pouze proměnné, čísla, parametry, řetězce pro porovnání a symboly || (or, disjunkce), && (and, konjunkce), <, >, ==,
- větev `then`, která se zavolá po splnění podmínky

  obsahuje slovo then a následně přechodovou funkci, nebo slovo `check` a název jiné podmínky a přidané parametry.
- Větev `else`, která jen místo slova `then` obsahuje slovo `else`, je totožná

  zavolá se, pokud podmínka není splněna.
- Na samostatném řádku znak }, který ukončuje podmínku.

Název parametrů může obsahovat pouze malá písmena, velká písmena, číslice, podtržítko _ a pomlčku -.

Ukázka použití. *Příklad reprezentující podmínku.*

```
check
   vA (x, y) {
   abeceda.last == nalozit
      then (x) jet (y)
      else (x) nalozit (x)
}
```

Vlastní výpis

Tato část je nepovinná pro běh programu. Slouží nám pro výpis vlastních informací do souboru. V souboru je vlastní výpis na jednom řádku z obou stran uzavřený znakem %. Mezi těmito znaky můžeme psát svůj text, ale můžeme využít i proměnné zmíněné výše.

Ukázka použití. *Příklad reprezentující vlastní výpis.*
%V case aktualni.cas proces proces.name je v aktualni.stav%

Příklad 5. *Příklad reprezentující převoz písku.*

```
co    pisek 1000 0 (A) (B) [(C)]
co    pisek { min=1; max=2; timeMin=150; timeMax=500; kapacitaMin=100;
kapacitaMax=200; from=[(A)]; to=[(C)] }
co    pisek { min=1; max=2; timeMin=150; timeMax=500; kapacitaMin=100;
kapacitaMax=200; road=[(A) (C) (B)] }

p     auto2 10 (C):100 10 10 10
p     auto { min=1; max=1; timeMin=0; timeMax=20; nosnostMin=25; nos-
nostMax=50; from=[(A)]; abeceda=[10-20 5-10 5] }

Q     (A)
Q     (B)
Q     (C)

F     Q

X     jet
X     nalozit + @
X     vylozit - @

d     (A) check vA [(A) (C)]
d     (C) check vC [(C) (A) (B)]
d     (B) check vB [(B) (C)]

check vA (x, y) {
   abeceda.last == nalozit
      then (x) jet (y)
      else (x) nalozit (x)
}
check vB (x, y) {
   abeceda.last == vylozit
      then (x) jet (y)
      else (x) vylozit (x)
}
check vC (x, y, z) {
   proces.name == auto
      then check vCauto [x, y, z]
      else check vCauto2 [x, y, z]
}
```

```
check vCauto (x, y, z) {
   abeceda.last == vylozit
      then (x) jet (y)
      else (x) vylozit (x)
}
check vCauto2 (x, y, z) {
   abeceda.last == nalozit
      then (x) jet (z)
      else (x) nalozit (x)
}

%Čas je aktualni.cas, proces proces.name je v aktualni.stav%
```

5.2 Gramatika

Vstupní data

Vstupní data pro běh gramatiky musí být uložen v souboru s příponou .txt, jiné koncovky program neakceptuje.

Vstupní soubor je rozdělen do několika částí:

- Proměnné
- Předměty
- Procesy
- Pravidla
- Cíle
- Podmínky
- Terminály
- Vlasní výpis

Komentáře v souboru jsou pouze jednořádkové, tj. od posloupnosti znaků `//` do konce řádku.

V programu bude gramatika převedena na konečný automat, který bude následně zpracováván.

Terminál je ekvivalentní abecedě z automatu. Název terminálu může obsahovat pouze malá písmena, velká písmena, číslice, kulaté závorky `()`, čárku `,`, podtržítko `_` a pomlčku `-`.

Neterminál je ekvivalentní stavu z automatu. Název neterminálu může obsahovat pouze malá písmena, velká písmena, číslice, kulaté závorky `()`, čárku `,`, podtržítko `_` a pomlčku `-`. Neterminál nemůže mít název Q.

Proměnné, Předměty, Procesy a Vlastní výpis jsou části gramatiky totožné s částmi z automatu. *Terminály* jsou totožné s částí *abeceda* z automatu.

Pravidla

Tato část je povinná pro běh programu. Udávají nám přechod z jednoho neterminálu do jiného pomocí terminálu.

Zápis pravidla v souboru:
řádek začíná znakem **d**, který je následován tabulátorem.
Dále může nastat jedna z následujících možností:

- jednoduché pravidlo
- pravidlo s podmínkou
- volání podmínky
- cílové pravidlo

Jednoduché pravidlo

Jednoduché pravidlo je pravidlo, které popisuje přechod z jednoho neterminálu do jiného. Toto pravidlo nepovoluje popis přechodu z neterminálu do neterminálu z množiny neterminálů.

Toto pravidlo obsahuje:

- neterminál, ze kterého je pravidlo voláno,
- soubor znaků −>,
- terminál, pomocí kterého se dostaneme do cílového neterminálu,
- neterminál, kam se přesuneme pomocí terminálu.

Ukázka použití. *Příklad reprezentující jednoduché pravidlo.*
d (vydejce,vylozen) −> nalozit (vydejce,nalozen)

Pravidlo s podmínkou

Pravidlo s podmínkou je pravidlo, které nám dovoluje si vybrat z více neterminálů.

Toto pravidlo obsahuje:

- neterminál, ze kterého je pravidlo voláno,
- soubor znaků −>,
- terminál, pomocí kterého se dostaneme do cílového neterminálu,
- neterminál, kam se přesuneme pomocí terminálu,
- v { } podmínka

  pokud je tato podmínka splněna, tak použijeme terminál a neterminál zmíněný výše,

  pokud ne, tak půjdeme na další řádek.
- Na dalším řádku je znak |, který slouží pro oddělení posloupností terminálů, neterminálů a podmínek,
- za tímto znakem následuje terminál a neterminál,
- za neterminálem může následovat další podmínka

  pokud zde bude, tak pravidlo musí pokračovat na dalším řádku stejně jako tento,

  pokud zde podmínka nebude, tak toto pravidlo končí.

Ukázka použití. *Příklad reprezentující pravidlo s podmínkou.*
d (patro1) −> in_ out (patro1) { predmet.in == (patro1) && proces.kapacita > 0 || (patro1) == predmet.out}
| nahoru (patro2) { predmet.in > (patro2) }
| dolu (patro0)

Volání podmínky

Toto pravidlo slouží k zavolání podmínky.

Toto pravidlo obsahuje:

- neterminál, ze kterého je pravidlo voláno,
- soubor znaků −>,
- slovo `check`, které nám říká, že budeme volat podmínku,
- název podmínky, která se bude volat,
- v [ ] oddělených mezerou seznam parametrů, které se předají podmínce.

Ukázka použití. *Příklad reprezentující volání podmínky.*
d (patro1) −> check podminka [(patro1) (patro2) (patro3)]

Cílové pravidlo

Toto pravidlo slouží k zapsání neterminálu, který slouží jako cíl.

Toto pravidlo obsahuje:

- neterminál, ze kterého je pravidlo voláno,
- soubor znaků −>,
- znak $, který nám říká, že neterminál bude i cíl

  tento znak lze použít pouze v cílovém pravidle, tzn. nelze použít ani v pravidle s podmínkou.

Ukázka použití. *Příklad reprezentující cílové pravidlo.*
d (prijemce,vylozen) −> $

Cíle

Tato část gramatiky je povinná. Je ekvivalentní k přijímacím stavům z automatu.

Cíl lze do souboru zapsat jednou z následujících možností:

- pomocí syntaxe přijímacích stavů z automatu

  F tab neterminál
- pomocí cílového pravidla

Podmínky

Podmínky mají stejnou funkci jako u automatu.
Tvar se liší pouze ve větvi `then`/`else`. Pokud větev volá jinou podmínku, tak se nic nemění. Pokud nevolá podmínku, tak je v souboru větev zapsána takto:

then/else terminál neterminál

Ukázka použití. *Příklad reprezentující podmínku.*

```
check
   podminka (x, y, z) {
   predmet.in == x && proces.kapacita > 0 || x == predmet.out
      then in_out (x)
      else check nah_dol [x, y, z] }
```

Příklad 6. *Příklad reprezentující převoz písku.*

```
co    pisek 1000 0 (vydejce,vylozen) (prijemce,nalozen)

p     auto 100 (vydejce,vylozen):0 10 10 30

F     (prijemce,vylozen)

X     nalozit +
X     vylozit -
X     jet

d     (vydejce,vylozen) -> nalozit (vydejce,nalozen)
d     (vydejce,nalozen) -> jet (prijemce,nalozen)
d     (prijemce,nalozen) -> vylozit (prijemce,vylozen)
d     (prijemce,vylozen) -> jet (vydejce,vylozen)
d     (prijemce,vylozen) -> $
```

5.3 Ovládání

Spuštění

Program po spuštění zobrazí okno se třemi tlačítky:

Nový - otevře prázdný editor pro psaní automatu.

Otevřít - otevře v editoru vybraný soubor s automatem,

- soubor lze upravovat i spouštět.

Zavřít - zavře celý program.

Editor

Editor slouží k úpravě automatu. Celé okno obsahuje textový editor, který slouží pro úpravu textu, checkbox, který povolí výpis do souboru, textové pole pro napsání cesty k souboru a menu se dvěmi záložkami:

- Soubor

  záložka dále obsahuje položky Nový, Otevřít, Uložit, Zavřít.

- Run

  záložka obsahuje položku Automat a Gramatika.

Menu - Soubor

V menu Soubor jsou následující položky:

- Nový

  uloží rozpracovanou část v editoru a následně editor vyprázdní,

  lze použít i zkratku `Ctrl + N`.

- Otevřít

  uloží rozpracovanou práci (pokud nějaká byla) a otevře dialog pro vybrání souboru s dříve rozpracovaným automatem,

  lze použít i zkratku `Ctrl + O`.

- Uložit

  uloží rozpracovanou práci, pokud práce nebyla již dříve uložena, nabídne dialog pro uložení práce z editoru,

  lze použít i zkratku `Ctrl + S`.

- Uložit jako ...

  vždy nabídne dialog pro uložení práce z editoru.

- Zavřít

  uloží práci a uzavře celý program.

Menu - Run

V menu Run je následující položka:

- Automat

  tato položka uloží rozpracovanou práci a spustí běh automatu.

- Gramatika

  tato položka spustí převod gramatiky na automat a spustí běh automatu.

Grafické zobrazení

Po spuštění automatu se zobrazí nová okna pro zobrazení automatu a bližší informace o simulaci.

V hlavním okně se zobrazí v obdélnících stavy s čekajícími předměty. Zobrazují se v nich procesy s posledním použitým slovem z abecedy.

V dalším okně se zobrazují bližší informace o procesech, jako je jeho náklad.

V posledním okně se zobrazují informace z kalendáře. V horní části je aktuální čas. Dále se zobrazí procesy z kalendáře spolu s časem, kdy bude dále pracovat.

Problémy

Při výskytu chyb se zobrazí chyby v novém okně a v editoru budou chyby podtržené červeně.
Mezi základní chyby, které se zobrazí, patří chyby v logických částech souboru:

- Proměnné

  chyba při nedodržení syntaxe pomocí . (tečky), která zajišťuje rozdělení proměnné na hlavní část a konkretizující část.

- Předmět

  chyba jak při nedodržení syntaxe, tak i vložení nečíselných znaků do číselných částí.

- Proces

  chyba při vložení nečíselných znaků do číselných částí, ale také při nedodržení syntaxe, hlavně při položce start, která se skládá ze dvou částí oddělených : (dvojtečkou).

- Stavy

  chyba, při použití mezery v názvu stavu, která není povolená.

- Přijímací stavy

  stejná chyba jako u stavů.

- Abeceda

  zde u abecedy může nastat chyba při nedodržení syntaxe,

  například mezera v názvu nebo použití nesprávného znaku pro bližší určení činnosti slova abecedy.

- Přechodové funkce

  nastat zde může chyba při nedodržení syntaxe,

  zde ale může nastat i chyba, která se nezobrazí,

  například pokud při psaní parametrů použijeme místo mezery jiný oddělovač, tím změníme název a při vyhodnocování podmínky se nám nebudou shodovat názvy a program může pracovat nesprávně.

- Podmínky

  zde by mohl nastat problém při nedodržení syntaxe.

Ukončení programu

Pro ukončení programu je nutné uzavřít editor. Uzavřeme-lií jen grafické zobrazení, editor zůstane stále otevřen. Při uzavření editoru se rozdělaná práce uloží do souboru a program se zavře.

Závěr

Cílem bylo vytvoření programu umožňujícího provádět diskrétní simulace různých jevů. To se také podařilo.

Úspěchem je, že jsme za použití konečného automatu a regulární gramatiky vyřešili univerzalitu, protože univerzalita byla hlavním problémem programu. Výhodou bylo vyřešení nedeterminismu konečného automatu, které neobsahuje zanedbání nedeterminismu. Dalším úspěchem je vytvoření jednochuché vizualizace, která bude lehce zobrazena i na menších obrazovkách.

Program by bylo možno dále rozšiřovat. Například místo konečného automatu by šlo implementovat zásobníkový automat. To by ale z našeho pohledu nebylo až tak významným zlepšením, jelikož již teď jsme schopni simulovat běžné činnosti. Zajímavějším rozšířením by zřejmě bylo statistické vyhodnocení výsledků. Toto lze dnes podniknout tak, že výstup simulace zpracujeme v jiném programu.

Seznam použité literatury

[1] *Discrete Event Simulations* (on-line) Chorvatsko: Sciyo, 2010. ISBN 978-953-307-115-2. URL: http://www.intechopen.com/books/discrete-event-simulations

[2] Teo Y. M. *Advanced Modeling and Simulation Techniques: studijní texty* (on-line) Singapur: National University of Singapore, 1999/2000. URL: http://www.comp.nus.edu.sg/~teoym/cs6205/9900/L1/sld001.htm

[3] Pidd, M. *Computer simulation in management science.* Velká Británie: John Wiley, 1992. ISBN 0-471-93462-3.

[4] Kočička, P. *Simulační metody jako nástroj pro rozhodování podniku - modelování pomocí programu Witness.* Brno: Masarykova univerzita, 2009.

[5] Malík, M. *Počítačová simulace.* Skripta MFF UK. Praha: UK Praha, 1989. ISBN 0-201-85449-X.

[6] Özgün, O., Barlas, Y. *Discrete vs. Continuous Simulation: When Does It Matter?* (on-line) Istanbul: Boğaziçi University, 2009. URL: http://www.systemdynamics.org/conferences/2009/proceed/papers/P1199.pdf

[7] Park, S., Leemis, L. *Discrete-Event Simulation: A First Course* The College of William and Mary, 1994, revision 2004. ISBN 0-13-202056-4.

[8] Jančar, P. *Teoretická informatika: učební text* (on-line). 1. vydání. Ostrava: Ediční středisko VŠB-TUO, 2007. ISBN 978-80-248-1487-2. URL: http://www.cs.vsb.cz/jancar/TJAA/tjaa_2p.pdf

[9] Kocour, P. *Úvod do teorie konečných automatů a formálních jazyků.* 1. vydání. Plzeň: Západočeská univerzita v Plzni, 2000. ISBN 80-7082-813-7.

[10] Vavrečková, Š. *Teorie jazyků a automatů: studijní texty* (on-line). Opava: Slezská univerzita v Opavě, 2007. URL: http://axpsu.fpf.slu.cz/~vav10ui/obsahy/tja/skripta/tja_celek.pdf

[11] Barták, R. *Automaty a gramatiky: studijní texty* (on-line). Praha: UK Praha, 2012. URL: http://ktiml.mff.cuni.cz/~bartak/automaty/lectures/all_lectures.pdf

[12] Černá, I., Křetínská, M., Kučera, A. *Automaty a formální jazyky I.* (on-line). Verze 1.3 Brno: Masarykova univerzita, 2002. URL: http://is.muni.cz/elportal/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf

[13] Hopcroft, J. E., Ullman, J. D. *Formálne jazyky a automaty.* Přel. B. Rovan, P. Mikulecký. 1. vydání Bratislava: Alfa, 1978. Přel. z: Formal languages and their relation to automata. ISBN 63-096-78

[14] *Internetové podpůrné prostředí k předmětu Teoretická informatika* Univerzita Hradec Králové: Fakulta informatiky a managementu. URL: `http://iris.uhk.cz/tein`

[15] MacDonald, M. *Pro WPF in C# 2010: Windows Presentation Foundation in .NET 4.0* USA: Apress, 2010. ISBN 978-1-4302-7205-2

[16] Richter, J. *CLR via C#* 3. vydání Washington: Microsoft Press, 2010. ISBN 978-0735627048

Seznam tabulek

Přílohy

Příloha A: Obsah přiloženého CD

Obsah přiloženého CD.

- Tato bakalářská práce v elektronické formě.
- Instalační soubor pro instalaci programu `Univerzální diskrétní simulátor`. Soubor se nachází v adresáři `Instalace`.
- Dokumentace programu v adresáři `Dokumentace`.
- Zdrojové kódy v adresáři `Zdrojové kódy`.
- Testovací data v adresáři `Test`.