FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z předmětu KIV/TI

Demonstrační program pro simulaci pevně zadaného nedeterministického rozpoznávacího automatu

Dominik Zappe (A20B0279P)

Jakub Mladý (A20B0190P)

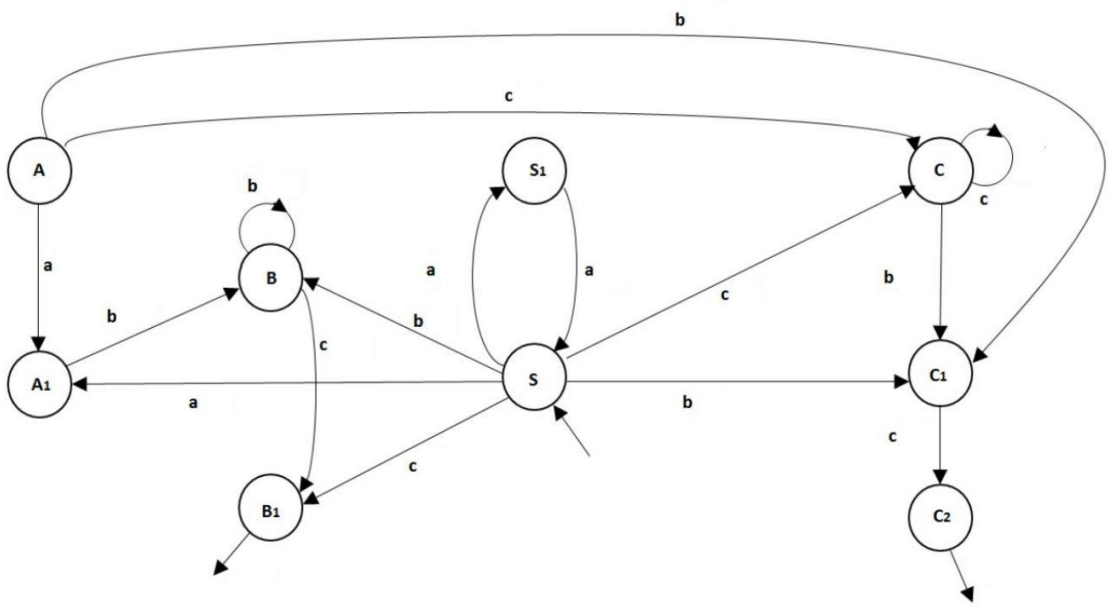
22. 11. 2021

Obsah

1. Zadání 1
2. Analýza úlohy 2
3. Automatový model 4
4. Implementace 4
4.1. Automat 4
4.2. Vizualizace 5
5. Uživatelská příručka 6
6. Závěr 8

1. Zadání

Z40. Demonstrační program pro simulaci pevně zadaného nedeterministického rozpoznávacího automatu



Obrázek 1: Přechodový graf zadaného nedeterministického rozpoznávacího automatu

Reprezentujte tento NKA přechodovým grafem. Pro zakreslení přechodového grafu použijte software JFLAP (https://www.jflap.org/)

Vytvořte program, který zobrazí přechodový graf s vybarvenými počátečními stavy. Program umožní postupně zadávat vstupní řetězec z klávesnice. Po zadání jednoho znaku program zobrazí přechodový graf s vybarvenými stavy, v nichž by se automat aktuálně mohl nacházet. Průběžně bude program zobrazovat informaci o tom, zda dosud zpracovaná část vstupního řetězce představuje akceptovaný řetězec či nikoli.

Kromě písmen vstupní abecedy program umožní také zpracovat tři speciální znaky, které (z demonstračních důvodů) umožní tyto akce: přechod „o krok zpět“, ukončení zpracování řetězce a zahájení zpracování nového řetězce, ukončení programu. Krok zpět může být aplikován i víckrát po sobě, až se dojde k počátečnímu stavu.

Důraz klaďte na grafickou stránku demonstrace, obrazovka by neměla rolovat.

2. Analýza úlohy

Z přednášek TI víme, že ke každému nedeterministickému rozpoznávacímu automatu existuje ekvivalentní deterministická verze. První krok je tedy převést si tento nedeterministický rozpoznávací automat na deterministický.

K tomu se dá buď využít ze zadání zmíněný software JFLAP, nebo se to dá také převést ručně. Nejprve si musíme znázornit automat pomocí tabulky (viz Tabulka 1).

Tabulka 1: Tabulka popisující přechodovou funkci automatu z Obrázku 1

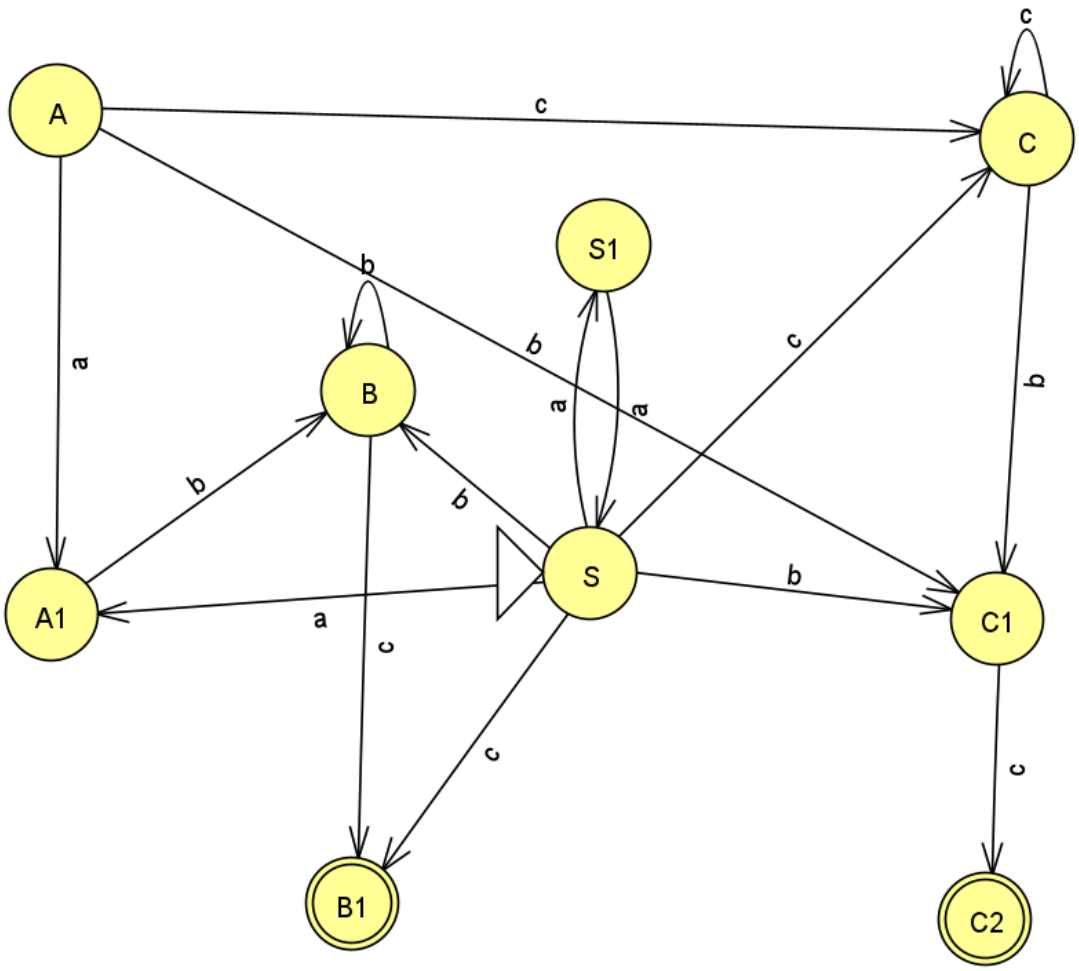
		a	b	c
→	S	{S1, A1}	{B, C1}	{C, B1}
	S1	{S}		
	A	{A1}	{C1}	{C}
	A1		{B}	
	B		{B}	{B1}
←	B1			
	C		{C1}	{C}
	C1			{C2}
←	C2			

Nyní vezmeme všechny nově vzniklé „stavy“ (množiny v jednotlivých sloupečcích) a vytvoříme novou tabulku, kde vstupní stav bude sjednocení nynějších vstupních stavů (v tomto případě je jen jeden, takže zůstane jeden). Nově vznikající stavy budou mít jako následující stavy v závislosti na vstupním symbolu opět množinu stavů, která vznikne sjednocením jednotlivých původních stavů (viz Tabulka 2). Jako výstupní stavy se pak označí takové množiny, které mají alespoň jeden výstupní původní stav.

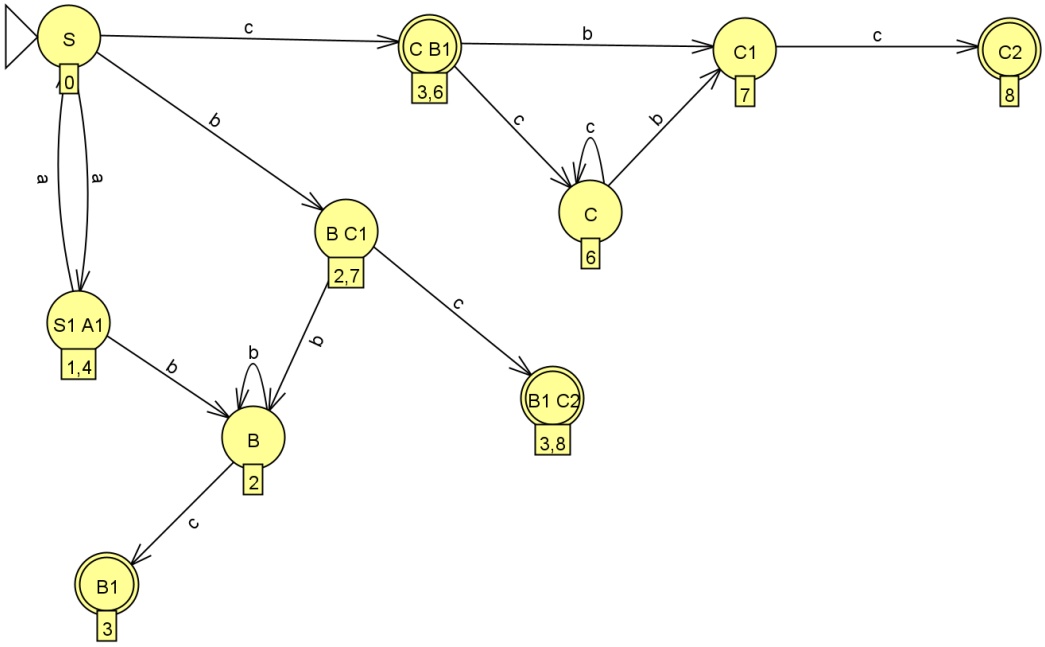
Tabulka 2: Tabulka ekvivalentního deterministického rozpoznávacího automatu

		a	b	c
→	{S}	{S1, A1}	{B, C1}	{C, B1}
	{S1, A1}	{S}	{B}	
	{B, C1}		{B}	{B1, C2}
←	{C, B1}		{C1}	{C}
	{B}		{B}	{B1}
←	{B1, C2}			
←	{B1}			
	{C}		{C1}	{C}
	{C1}			{C2}
←	{C2}			

Ručně vytvořené tabulky můžeme ověřit s výsledkem ze softwaru JFLAP. Následující obrázek (2) zobrazuje zadaný vstupní nedeterministický konečný automat (ze zadání) do JFLAPu. Obrázek 3 demonstruje výstup z JFLAPu, převedení na deterministický konečný automat. Jednoduše se tak dá ověřit správnost výše uvedené tabulky (2).



Obrázek 2: Nedeterministický rozpoznávací automat (ze zadání) v JFLAPu



Obrázek 3: Vygenerovaný ekvivalentní deterministický rozpoznávací automat pomocí JFLAPu

Nyní když už je nám znám deterministický ekvivalentní rozpoznávací automat se dá tato úloha jednoduše naimplementovat. Na pozadí se bude pracovat s deterministickou verzí a uživateli se bude vykreslovat původní nedeterministická. Šikovná pojmenování jednotlivých stavů výrazně zjednoduší budoucí vizualizaci v uživatelském rozhraní, např.: stav pojmenovaný „S1 A1“ nám jasně říká, že když se automat bude nacházet v tomto stavu, tak se uživateli má zvýraznit jak stav S1, tak stav A1.

3. Automatový model

Popis formou tabulek už byl výše zmíněn v analýze problému. Tabulka 1 popisuje původní nedeterministický rozpoznávací automat. Tabulka 2 popisuje ekvivalentní deterministickou verzi tohoto automatu.

Přechodové grafy obou automatů již byly také výše zmíněny. Přechodový graf nedeterministického rozpoznávacího automaty je již v zadání (Obrázek 1), ale je vidět i jeho JFLAP verze na obrázku 2. Obrázek 3 ukazuje přechodový graf ekvivalentní deterministické verze. Obrázky 2 a 3 jsou ze softwaru JFLAP, vstupní stavy jsou označené velkým černě ohraničeným bílým trojúhelníkem. Výstupní stavy tento software označuje kulatým zvýrazněním stavu poblíž ohraničení. Přechody mezi jednotlivými stavy jsou normálně zaznamenávány šipkami s popiskem, který určuje vstupní symbol.

4. Implementace

Implementace byla realizována v Javě. Nebylo využito žádných speciálních knihoven, pouze standardní knihovny Javy.

4.1. Automat

Obecně při implementaci konečných automatů se využívá výčtový typ, který slouží k pojmenování stavů. Důležitá je přechodová tabulka, která se obvykle implementuje jako dvourozměrné pole. Přechodová tabulka je typu výše uvedeného výčtového typu stavů. Neméně důležitá je výstupní tabulka určující výstupní funkci automatu. Implementace je obdobná jako u přechodové tabulky.

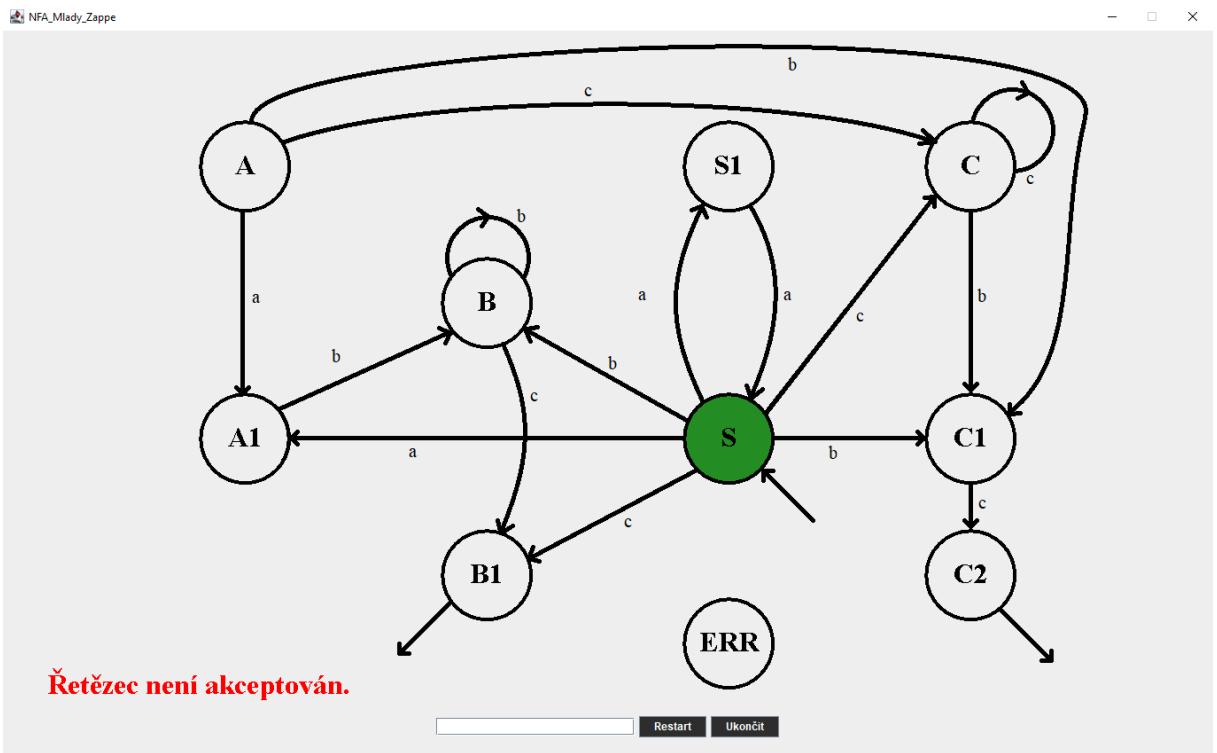
Tato práce pojala implementaci jinak. Bylo využito velmi objektově orientovaného přístupu k problému. První vytvořená třída reprezentuje deterministický konečný automat. Tato třída obsahuje vnitřní výčtový typ stavů. Možné stavy v tomto výčtu přímo odpovídají obrázku 3, popř. tabulce 2. Dále každý stav obsahuje přímo při inicializaci konstantní atribut typu boolean, říkající o stavu, zdali je výstupní nebo ne, např.: stav S má tento atribut nastaven na *False*, kdežto stavy *B1, C2, C B1* a *B1 C2* jsou nastaveny na *True*. Každý stav má dále další tři atributy – *a, b, c* – typu tohoto výčtového typu. Jednoduše se tak dá objektově zbavit přechodové tabulky a každému stavu se v následujícím statickém bloku nastaví hodnota daných atributů, např.: *S.a = S1A1, S.b = BC1* a *S.c = CB1*. Nyní je potřeba zavést funkci pro přechod mezi stavy, neboli když máme z venku výčtového typu nastaven nějaký stav na nynější, tak chceme, aby tato funkce vracela následující stav na základě vstupního symbolu. Toho se jednoduše docílí přes již zmíněné atributy *a, b, c*. Třída obsahující tento výčtový typ obsahuje dva atributy. První reprezentuje nynější stav a je tedy typu výše zmíněného výčtového typu. Druhý atribut je zásobník natypovaný na výčtový typ stavů. Tento zásobník pouze slouží pro funkci krokování zpět. Třída obsahuje tři funkce – *restart(), accept()* a *stepBack()*. Funkce *restart()* nastaví nynější stav na počáteční stav a vyprázdní zásobník. Funkce *accept()* přijme vstupní symbol jako parametr typu char. Do zásobníku se nejdříve pushne nynější stav a ten se dále nastaví na následující stav díky vnitřní funkci výčtového typu. Dále tato funkce vrací *true* nebo *false* na základě toho, jestli je nynější nový stav výstupní nebo ne. Poslední funkcí této třídy je *stepBack()*, která zajišťuje možnost krokovat zpět. Krokování zpět zajišťuje zásobník typu výčtového typu stavů. Pokud není prázdný, nastaví se nynější stav na předchozí stav, neboli provede se pop nad zásobníkem. Obdobně jako funkce *accept()* tato funkce vrací *true* nebo *false* na základě toho, jestli je nynější nově nastavený stav výstupní nebo ne.

4.2. Vizualizace

Vnitřní reprezentace a implementace automatu by nyní měla být jasná. Dalším krokem bylo vytvořit vizualizaci v okně. Hlavní vstupní bod programu – metoda *main()* – tedy vytváří okno a vkládá do něj dvě komponenty. Obě tyto komponenty dědí od třídy JPanel. První panel představuje vykreslování přechodového grafu pro nedeterministický rozpoznávací automat. Celé vykreslování se děje online. Jediný potřebný přídavný soubor je background.png představující obrázek na pozadí. Na tomto obrázku jsou pouze šipky s popisky. Vykreslování a znázorňování stavů se děje za běhu programu podle požadavků od uživatele. Druhý panel je vespod okna a je na něm vstupní textové pole, do kterého uživatel může vkládat jednotlivé znaky a tvořit tím vstupní řetězec. Nad tímto textovým polem byl inicializován posluchač, který hlídá změny. Když zaregistruje přidání symbolu, pošle tento přidaný symbol do deterministického automatu, který běží na pozadí programu, který byl již výše popsán. Změna stavu v deterministickém automatu se projeví na vizualizaci změnou označení možných stavů, v nichž by se mohl nedeterministický automat nacházet. Když posluchač nad textovým polem zaregistruje odstranění znaku, zavolá se funkce pro vrácení se o krok zpět (využití dříve zmíněného zásobníku). Dále se na tomto druhém panelu nachází dvě tlačítka. Jejich název jasně udává jejich funkci. Tlačítko restart restartuje automat a vizualizaci do počátečních stavů a smaže obsah textového pole. Tlačítko ukončit ukončuje běh programu.

Zvýraznění nynějšího stavu je implementováno přes vlastní objekt reprezentující jednotlivé graficky vyobrazené stavy. Tedy každý stav z výčtového typu má svůj vlastní objekt reprezentující jejich podobu na kreslícím panelu. Tato třída obsahuje základní informace pro vykreslení jako X, Y pozice v okně, velikost kolečka reprezentující stav nebo jeho název, který se vykresluje. Dále také obsahuje atribut určující, zda je tento stav nynější, či není; zda má být zvýrazněn ve vizualizaci, či nikoliv. Tato proměnná se nastavuje zvenčí, když dochází k jednotlivým přechodům mezi stavy.

Popsaná vizualizace je vidět na obrázku 4.



Obrázek 4: Vizualizace a výstup práce

5. Uživatelská příručka

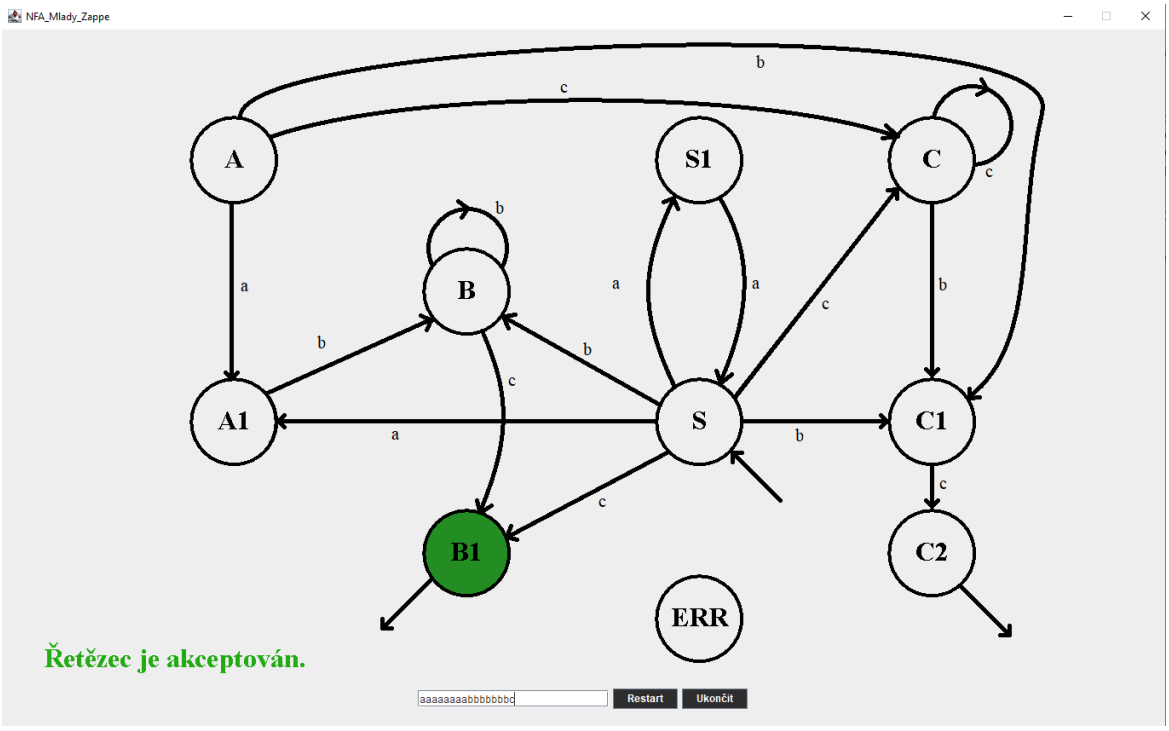
Pro zapnutí programu je potřeba mít nainstalovanou Javu verze 8 a výše. Dále je třeba do příkazové řádky (uživatel se musí nacházet v adresáři s .jar souborem) napsat příkaz java -jar nazev_souboru.jar. Tedy konkrétně java -jar SP_TI_Zappe_Mlady.jar. V adresáři s .jar souborem by se měl také nacházet soubor obrázku na pozadí – background.png. V adresáři je přiložen dávkový soubor run.cmd, který vykoná výše zmíněný příkaz z příkazové řádky sám. Stačí tak dvakrát kliknout na run.cmd.

Hned po zapnutí se načte okno a přes něj se vykreslí dialogové okno. V tomto dialogu je stručný popis toho, co tento program reprezentuje (Tento program znázorňuje výpočty, kterými může být zpracován vstupní řetězec nedeterministickým konečným automatem. Zvýrazněné stavy znázorňují množinu stavů, v jednom z nichž se v dané fázi zpracování řetězce automat bude nacházet (ale nelze určit ve kterém). Řetězec je automatem akceptován, může-li být po zpracování řetězce v koncovém stavu, tedy je-li po zpracování řetězce zvýrazněn alespoň jeden koncový stav.)

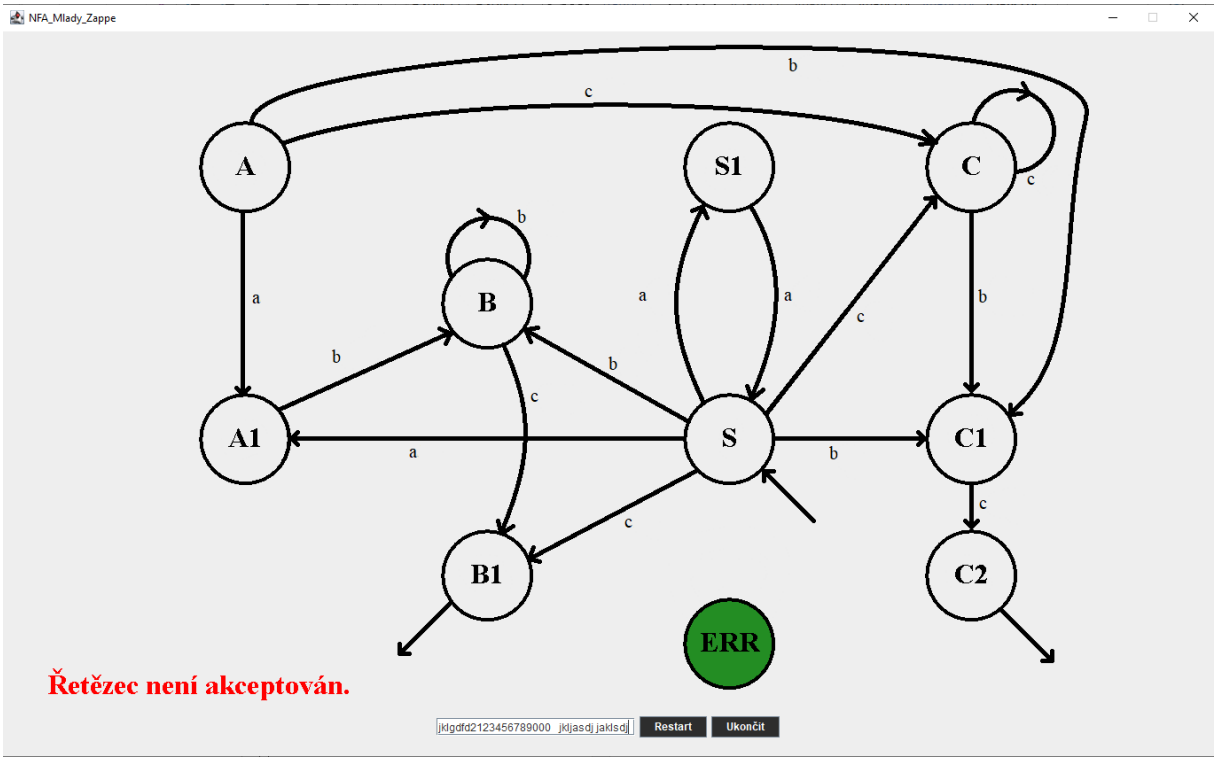
Po zavření dialogu by měl uživatel vidět okno s přechodovým grafem a zeleně označeným počátečním stavem S. Pod tímto přechodovým grafem se v okně nachází textové pole, do kterého uživatel může psát libovolné znaky. Vstupní symboly abecedy jsou *a*, *b* a *c*. Pokud uživatel zadá jakýkoliv jiný, neznámý, znak, automat skončí ve stavu ERR představujícím absorpční neakceptující stav. Do stavu ERR se uživatel dostane vždy, když zadá symbol, který z daného stavu nemá žádný následující stav tímto symbolem. Tedy z každého stavu by do stavu ERR mělo vést tolik hran, kolik je písmen v abecedě, potažmo i číslic atd. Pro jednoduchost vizualizace jsou tyto hrany „zanedbány“. Z textového pole lze libovolně znaky mazat pro krokování zpět. Vedle textového pole se nachází dvě tlačítka – restart a ukončit. Tlačítko restart vrátí přechodový graf do původního počátečního stavu a smaže obsah textového pole. Tlačítko ukončit ukončuje běh programu.

Vlevo vedle textového pole pod přechodovým grafem je buď červeně napsáno, že zadaný řetězec není akceptován tímto automatem, nebo zeleně napsáno, že zadaný řetězec je akceptován.

Obrázek 5 demonstruje příklad akceptovaného řetězce. Obrázek 6 demonstruje funkčnost chybového absorpčního stavu. Na těchto obrázcích si také lze povšimnout zmíněných tlačítek.



Obrázek 5: Příklad akceptovaného řetězce (aaaaaaaabbbbbbbc)



Obrázek 6: Příklad neakceptovaného řetězce a ukázka použití symbolů, které nejsou ve vstupní abecedě

6. Závěr

Hlavním kladem naší práce je objektově orientovaný přístup k implementaci automatu. Jedná se o zcela odlišný pohled na věc a šetří se paměť, neboť není nutné deklarovat 2D pole pro přechodové a další funkce. U tak malého počtu stavů to samozřejmě nijak neovlivňuje složitost programu, ale u složitějších automatů by tomu tak mohlo být.

Vylepšení naší práce by mohla být hezčí vizualizace uživatelského prostředí. Samozřejmě záleží na subjektivním posouzení, my jsme s vizualizací spokojeni.

Toto téma není nijak možné rozšířit, neboť byl vstupní nedeterministický automat pevně zadán, ale naše implementace by šla rozšířit o více stavů a větší vstupní abecedu.