Samostatná práce z předmětu KIV/TI

Automat na řízení křižovatky

Jakub Vítek, A16B0165P
Filip Hanzelín, A16B0037P

V Plzni dne 5. 1. 2017

1 Obsah

2 Semestrální práce

2.1 Zadání

Úkolem této semestrální práce je navrhnout a simulovat konečný automat, díky kterému bude možné řídit existující křižovatku “U divadla Alfa” v Plzni.

2.1.1 Analýza úlohy

V rámci úlohy se snažíme vytvořit simulaci řízení křižovatky o 15 fyzických semaforech a dvou fyzických přechodů pro chodce. Řízení jednotlivých částí křižovatky nemusíme řešit zvlášť, můžeme některé semafory sjednotit do skupin, díky čemuž jsme schopni redukovat výsledný počet stavů v konečném automatu.

Na základě toho, ve kterém pořadí chceme měnit rozsvícení signálů na jednotlivých semaforech jsme schopni pro náš konečný automat vytvořit stavový graf, ve kterém vrcholy budou představovat jednotlivé stavy, do kterých se může automat dostat, a hrany s ohodnocením, které definují to, do jakého stavu se automat přesune po přečtení vstupu určeného právě tím ohodnocením.

V rámci simulace je nejvýše vhodné využívat specificky používané makro stavy. U těchto stavů je třeba vždy definovat posloupnost akcí, která zajistí změnu z předchozího stavu na stav aktuální, přičemž akce se budou provádět ve chvíli, kdy se přesuneme z předchozího stavu do aktuálního. Ve výsledku se tedy jedná o specifické využití Mooreova typu konečného automatu.

2.1.2 Datové struktury

Pro simulaci je nutné ukládat základní informace, které nám definují konečný automat. Při návrhu aplikace jsme vycházeli z matematické definice konečného automatu:

$A = \{Q, \Sigma, O, \delta, q_{0,} \lambda\}$

U množiny stavů, množiny vstupů a množiny výstupů je dostatečné realizovat jejich uložení za pomocí datové struktury ArrayList. Již i obyčejné pole by pro realizaci bylo dostatečné, tato datová struktura za nás však řeší problém toho, že při obecné implementaci konečného automatu tak, aby byl maximálně nastavitelný nemůžeme s jistotou říci, kolik prvků by výsledné pole mělo.

Dále potřebujeme určit datovou strukturu, do které uložíme námi navržený stavový strom. Jedná se tedy o objekt z teorie grafů - orientovaný graf. Ten je definován následujícím způsobem:

G = <V, E>
V - neprázdná množina vrcholů (stavů konečného automatu)
E - množina uspořádaných dvojic vrcholů grafu (tzn. $E \subseteq V \times V$)

Z definice orientovaného grafu a z omezení programovacího jazyka Java vyplývá, že budeme muset vybrat datovou strukturu, která nám do sebe dovolí ukládat dvojice vrchol a množina hran, u kterých hrana začíná ve vrcholu určeného prvním členem dvojice. Celý myšlenkový postup směřuje k následující definici:

HashMap<key, value> => HashMap<v, ArrayList<e>>

Tuto definici můžeme přeložit jako “Pro každý vrchol **v** máme list hran, který obsahuje nespecifikované množství hran **e**”. V případě datové struktury HashMap musí být veškeré klíče unikátní, jinak bychom mohli do datové struktury přidávat dvojice typu $\{v_a, e_1\}$, $\{v_a, e_2\}$, což by nám k vrcholu s označením a přidalo hrany s označeními 1,2.

V tomto případě musíme ukládat data tak, že pro každý vrchol v_a máme množinu hran. Výsledná dvojice tedy bude vypadat následujícím způsobem: $\{v_a, \{e_1, e_2\}\}$.

2.2 Implementace

2.2.1 Třída Main

2.2.1.1 metoda main

Metoda, která je volána při spuštění aplikace. Ta spouští metodu setup, která dle určitých pravidel (definovaných v té dané metodě) připraví vnitřní stav objektu konečného automatu (Automaton.java) a následně připravenou strukturu předá části aplikace, která řeší grafický výstup.

2.2.1.2 metoda setup

Nejprve musíme vytvořit samotný objekt konečného automatu.

Automaton automat = Automaton.getInstance(); - *v aplikaci pouze jediný automat*

Před použitím konečného automatu je nutné definovat jeho vnitřní strukturu. To znamená, že se za pomocí metod objektu musí nadefinovat jeho vnitřní stav. Metoda musí v daném pořadí registrovat všechny vstupy (znaky) na které bude konečný automat reagovat.

automat.addInput('c'); - *automat bude reagovat na vstup c*

Následně je nutné přidat počáteční stav a rovnou jej jako počáteční v automatu označit. Následně je nutné přidat zbývající stavy a po jejich přidání některé z nich označit jako stavy koncové (je-li to nutné).

State s0 = new State("0"); - vytvoříme počáteční stav
automat.setStart(s0); - označíme počáteční stav

State s1 = new State("1"); - vytvoříme a přidáme libovolné možství stavů
State s2 = new State("2");
automat.addState(s1);
automat.addState(s2);

Neposledně musíme do automatu přidat všechny hrany mezi jednotlivými stavy. Hrany se definují

Edge s0c = new Edge(s0, s1, 'c'); - hrana ze stavu s0 do stavu s1 vstupem c
Edge s1c = new Edge(s1, s2, 'c');
Edge s2c = new Edge(s2, s0, 'c');
automat.addEdge(s0c); - samozřejmě hrany musíme přidat do automatu
automat.addEdge(s1c);
automat.addEdge(s2c);

Poslední volitelnou částí simulovaného konečného automatu je možnost nastavení automatického časovaného vstupu:

Automaton.getInstance().setTimedInput('c', 5000); - zavolá vstup/hranu ‘c’ každých 5 sekund

2.2.2 Třída State

Tato třída reprezentuje jeden stav v konečném stavovém automatu. Každý stav má svůj textový identifikátor. U instance této třídy pak máme getter pro získání daného identifikátoru, metodu pro převod dané instance na textovou podobu a metodu macro.

2.2.2.1 metoda macro

Překrytím této metody při vytváření instance je možné změnit výslednou instanci na makro stav konečného automatu. Samotný automat pak zajistí aby všechny příkazy obsažené v této metodě byly zavolány ve chvíli, kdy automat do tohoto stavu přejde. Makro stav se definuje následujícím způsobem:

```
State macro = new State(“macro”)) {
        @Override
        public void macro() {
                System.out.println(“Automat se dostal do stavu ” + this.getID());
        }
};
```

2.2.3 Třída Edge

Tato třída odpovídá objektovému návrhovému vzoru přepravka. Při tvorbě instance této třídy je nutné specifikovat zdrojový a cílový stav konečného automatu a vstup pro který je tato hrana definována, například new Edge(source, target, ‘a’) a new Edge(source, target, ‘b’) jsou rozdílné hrany v konečném automatu. Lze také definovat hranu, která začíná a končí ve stejném stavu.

Třída také mimojiné obsahuje příslušné gettery ke všem hodnotám, které jsou do ní ukládány a metodu pro převod instance a textovou hodnotu.

2.2.4 Třída TimedInput

Třída, která zajišťuje periodické vysílání vstupu do konečného automatu při simulaci. Instanci je nutné definovat na základě vstupu, který bude generován a času, za který se má vstup opakovat.

2.2.4.1 metoda run

Vytvoří nový Timer objekt, který bude pravidelně opakovat vkládání vstupu do konečného automatu na základě zvoleného času.

2.2.4.2 metoda stop

Zruší pravidelné opakování vkládání vstupu do konečného automatu v případě, že je takové opakování aktivní.

2.2.5 Třída Automaton

Tato třída reprezentuje simulovaný konečný automat. Obsahuje základní metody, které dovolují přidání nutných částí konečného automatu (dle jeho matematické definice viz. Analýza úlohy). Následně obsahuje metody pro zpracování vstupního řetězce konečného automatu. Dalé třída obsahuje možnost určit, zda lze konečný automat simulovat či zda je možné pouze měnit jeho strukturu.

2.2.5.1 Status automatu

Automat může být ve dvou stavech. Tyto stavy byly při implementaci vytvořeny z toho důvodu, aby při simulaci nebylo možné editovat strukturu a stavový graf tak, že by se automat mohl ve výsledku dostat do nepředpokládaného stavu.

Stav automatu je určován proměnnou running, pokud je nastavena na false, jedná se o stav automatu, ve kterém lze měnit jeho strukturu, přidávat nové hrany a vrcholy do stavového grafu. Pokud je však hodnota nastavena na true, do automatu nelze přidávat nové prvky - lze jej pouze simulovat.

V rámci svého běhu se automat může dostat do chybového stavu, například při použití vstupu, který nebyl povolen při definici automatu. Samotný chybový stav je absorbční, nelze se z něj dostat použitím jakéhokoliv dalšího stavu. Prakticky se jedná o signalizaci toho, že struktura automatu či stavový strom byly špatně navrženy nebo byly nekompletní.

2.2.5.2 Zpracování vstupu při simulaci

Současná simulace konečného automatu vždy reaguje na vstup, který je určen jedním písmenkem v abecedě. Simulátor automatu si vždy drží informaci o tom, ve kterém stavu se simulovaný konečný automat momentálně nachází.

Na základě vstupu se prohledají všechny hrany konečného automatu, které začínají v aktuálním vrcholu/stavu konečného automatu a využívají ten daný vstup. Je nutné doplnit, že zpracování vstupu předpokládá, že výsledná struktura/stavový graf reprezentují deterministický konečný automat. To znamená.že z jednoho vrcholu pro jeden vstup budeme mít pouze jednu hranu - tato skutečnost je hlídána při vytváření struktury/stavového grafu před spuštěním simulace automatu.

Jako vstup automatu je možné použít celý řetězec platných vstupů, který se bude zpracovávat znak po znaku (metody tick(String s) => tick(Character c)). Postup zpracování vstupního znaku je vždy stejný.

Pokud je automat ve stavu simulace, nedostali jsme se do chybového stavu a zpracovávaný vstup je platným vstupem, ověřujeme zda pro vrchol/stav, ve kterém je momentálně konečný automat, existuje hrana, díky které je možné dostat se z aktuálního stavu daným vstupem do cílového stavu určeného hranou konečného automatu.

V případě, že žádná taková hrana neexistuje, automat je přesunut do chybového absorpčního stavu. V případě, že je možné se přesunout hranou do dalšího stavu, změníme aktuální stav, ve kterém se nachází automat pro simulátor na stav, do kterého jsme se hranou přemístili.

Následně provedeme metodu macro toho stavu/vrcholu, do kterého jsme se přesunuli. Nutno doplnit, že stavy/vrcholy mají vždy mají minimálně metodu macro prázdnou (v tomto případě se nepočítá daný stav mezi makro stavy).

Posledním krokem nepovinným zpracování může být zapnutí časovaného vstupu, který automaticky zavolá metodu zpracování automatického vstupu.

2.2.6 Ostatní třídy

Aplikace obsahuje další pomocné třídy, které nejsou pro samotný běh aplikace zajímavé. Většinou se jedná o knihovní třídy, jejiž metody nebylo moudré implementovat do tříd vypsaných výše. Dále zde jsou také dvě třídy, které využívají java framework JavaFX a neřeší nic jiného, než zobrazení výstupu uživateli. Pro úplnost zde vypisujeme dané třídy, nebudeme je však popisovat konkrétněji.

- GUI
    - Hlavní třída pro JavaFX, díky ní je voláno vytváření struktury GUI. Vzhledem k tomu, že JavaFX blokuje hlavní vlákno až do ukončení vykreslovaného okna, tato třída také spouští simulaci automatu do nového vlákna.
- MainView
    - Zde se vytváří struktura okna, například tlačítka a reakce na ně či rozmístění jednotlivých prvků.
- ImageReference
    - Prázdná třída, která slouží pouze pro přístup k obrázkům uloženým ve stejném balíku v případě, že je aplikace distribuována jako spustitelný jar soubor (tj. vždy).
- Logger
    - Třída, která zjednodušeně zajišťuje stejnou funkci, jako například známka knihovna log4j2, tj. umožňuje formátování a výstup logovacích zpráv.
- TextConsole

    - Upravená JavaFX TextArea, do které je možné přesměrovat proud OUT pro vytvoření jednoduché konzole pro čtení přímo v aplikaci.
- TextNormalizer
    - Třída, která zajišťuje úpravu logované zprávy před jejím odesláním do výstupu loggeru.

2.3 Uživatelská příručka

2.3.1 Spuštění aplikace

- Aplikaci je možné spustit dvouklikem na distribuovaný jar.
- Aplikaci je možné spustit z příkazové řádky příkazem:
    - java -jar kiv-ti-sp-vitek-hanzelin-R2.jar

2.3.2 Rozložení aplikace

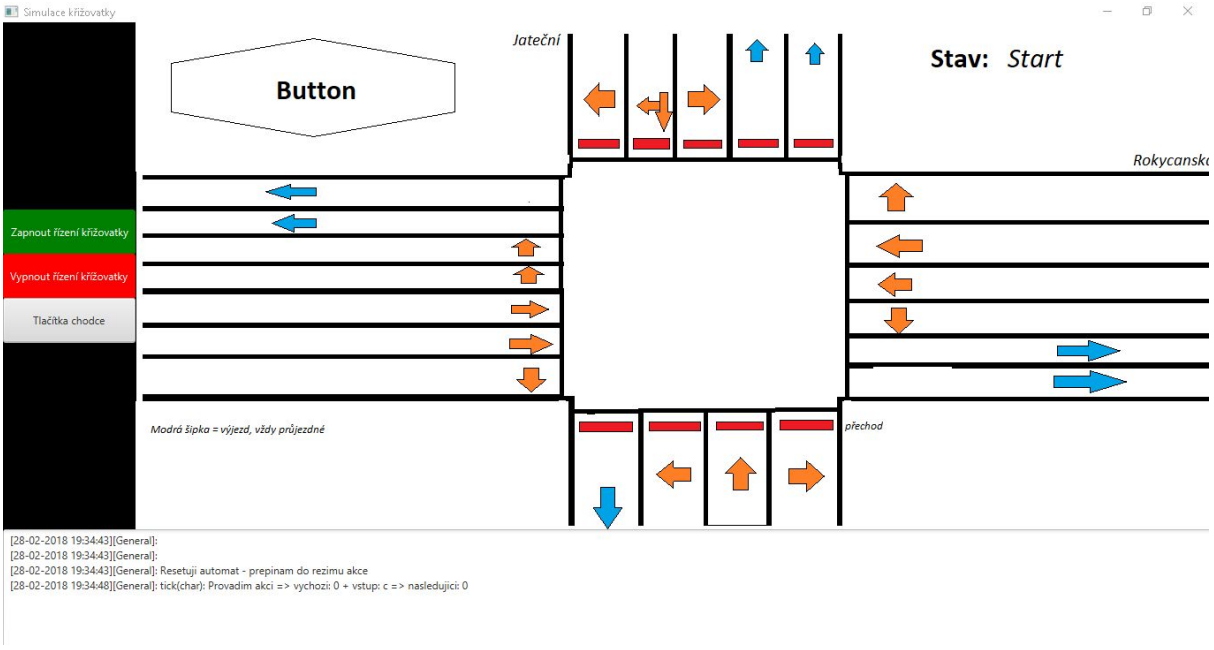
Grafická podoba aplikace je rozdělena do tří částí. Levá horní část obsahuje ovládací tlačítka. Tlačítko “Zapnout řízení křižovatky” vyresetuje konečný automat, převede jej do počátečního stavu a nastaví automatický vstup.

Tlačítko “Vypnout řízení křižovat” převede křižovatku do stavu, ve kterém všechny semafory budou svítit oranžově. Křižovatka tak nebude řízena.

Tlačítko “Tlačítko chodce” simuluje stav, kdy bylo stisknuto jakékoliv z tlačítek pro chodce. Automat se v případě nutnosti dostane na alternativní “okruh”, který lépe vyhoví chodci.

V pravé horní části je pak vykreslen aktuální stav křižovatky. Vždy přístupné proudy silnice jsou označeny modře. Ostatní barvy šipek pak reprezentují aktuálně rozsvícená světla jednotlivých semaforů. V případě nutnosti je tato část doplněna dalšími informacemi. Pokud bylo již zmáčknuto tlačítko chodce, vybarví se oblast s názvem Button červeně.

Spodní část obsahuje jednoduchý výpis z logování aplikace. Je zde možné vidět v textové podobě jakým způsobem probíhá fungování aplikace.



3 Závěr

V rámci této semestrální práce jsme vytvořili funkční způsob, jak řídit již existující křižovatku v Plzni. Při vývoji této práce jsme vytvořili systém, který nám dovolí v naší aplikaci definovat a simulovat jakýkoliv jednoduchý deterministický konečný automat. Případě že bychom tedy konečný automat jako takový chtěli změnit bylo by to velice snadné - stejný případ bohužel neplatí pro samotnou vizualizace, jejíž předělání by v současném stavu poněkud obtížnější.