# SEMESTRÁLNÍ PRÁCE Z PŘEDMĚTU KIV/TI

K9. Demonstrační program pro simulaci nedeterministického konečného automatu

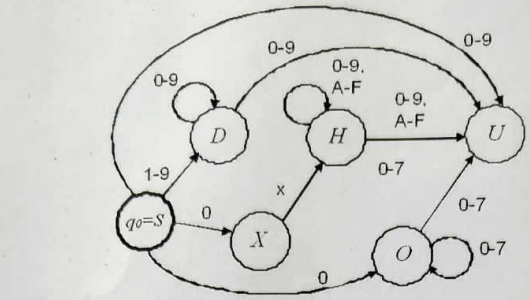
13. června 2024
Filip Valtr – A22B0107P
Václav Prokop – A22B0330P

# Obsah

# Zadání

**K9. Demonstrační program pro simulaci nedeterministického konečného automatu**

Je dán rozpoznávací NKA




S počáteční stav U koncový stav

Vytvořte demonstrační program, který zobrazí přechodový graf s vybarveným počátečním stavem. Program umožní postupně zadávat vstupní řetězec z klávesnice. Po zadání každého 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 dva speciální znaky, které umožní ukončení zpracování řetězce a zahájení zpracování nového řetězce (RESET), respektive ukončení programu (STOP).

Důraz klaďte na grafickou stránku demonstrace.

Rady:

Nejprve k automatu („ručně“) nalezněte ekvivalentní deterministický automat.

Na http://home.zcu.cz/~vais/ v rozšiřujícím materiálu o konečných automatech prostudujte kapitolu Principy softwarové implementace.

Jeden z možných přístupů k řešení: V nějakém editoru si připravte sadu obrázků. Na každém z nich budou vybarveny stavy, které odpovídají jednomu stavu deterministického ekvivalentu. Takový obrázek udělejte pro každý stav deterministického ekvivalentu. Potom výstupní akce programu související s přechodem bude spočívat pouze ve výměně obrázku. Obrazovka by neměla rolovat.

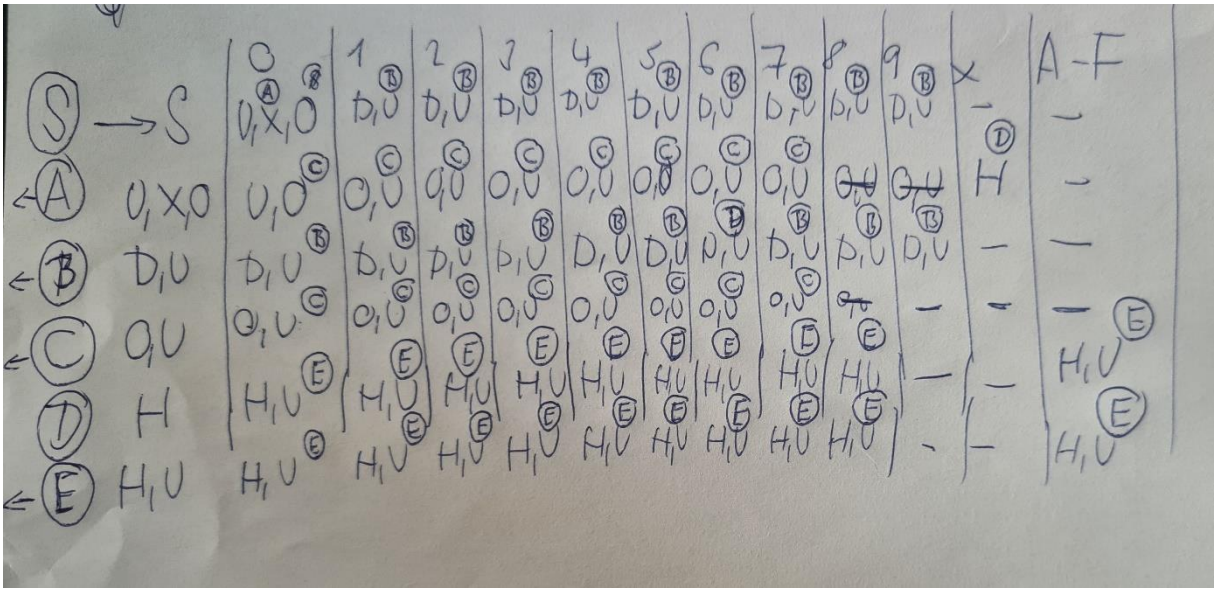
*Zadání projektu 1*

## Analýza úlohy

Jako první jsme se rozhodli z rozpoznávacího nedeterministického automatu naleznout ekvivalentní deterministický automat. Víme, že počáteční stav deterministického konečného automatu odpovídá množině počátečních stavů nedeterministického automatu, a navíc KA vytváříme tak, že krok po kroku vyhodnocujeme přechodovou funkci δ. Začínáme od počátečního stavu, tedy S.

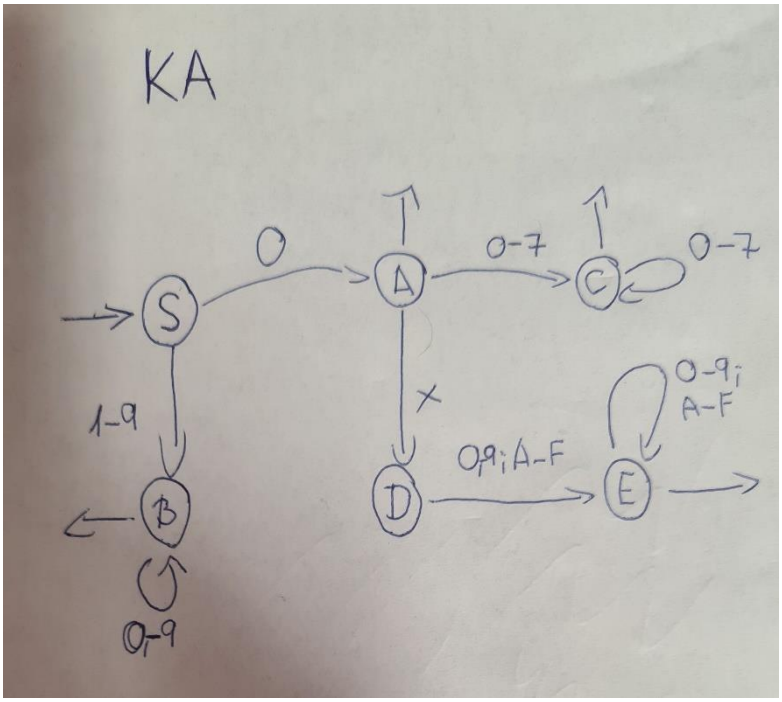
Stavy nedeterministického automatu nakonec sjednotíme do jednoho stavu deterministického automatu, s tím, že navíc platí, že množinou koncových stavů deterministického automatu budou všechny stavy, které v sobě obsahují některý z koncových stavů ekvivalentního nedeterministického konečného automatu.

Tabulka pro nález ekvivalentního deterministického konečného automatu bude tedy vypadat takto:



*Tabulka 1*

Nyní za pomocí tabulky vytvoříme deterministický konečný automat, ten bude vypadat následovně:




*Deterministický konečný automat 1*

Dále využijeme sady obrázků, kde bude každý stav deterministického automatu vyobrazen obrázkem, který odpovídá ekvivalentnímu nedeterministickému stavu automatu. Navíc bude v obrázku vyobrazeno, zdali je daný řetězec akceptován, tedy nachází-li se v nějakém z koncových stavů, či nikoliv.

Platí, že pokud v některém stavu přijde vstupní symbol, pro který v přechodovém grafu neexistuje hrana, pak to znamená, že symbol není automatem zpracovatelný, je tedy automatem zamítnut, což se v naší aplikaci projeví vypnutím aplikace a vypsáním chybové hlášky.

## Automatový model

-popis vstupních a výstupních signálů formou tabulek

| Stav | Vstup | Výstup |
|---|---|---|
| S | 0 | Je akceptován |
| S | 1-9 | Je akceptován |
| A | X | Není akceptován |
| A | 0-7 | Je akceptován |
| B | 0-9 | Je akceptován |
| C | 0-7 | Je akceptován |
| D | 0-9,A-F | Je akceptován |
| E | 0-9,A-F | Je akceptován |

| Stav | Vstup | Následující stav |
|---|---|---|
| S | 0 | A |
| S | 1-9 | B |
| A | X | D |
| A | 0-7 | C |
| B | 0-9 | B |
| C | 0-7 | C |
| D | 0-9, A-F | E |
| E | 0-9, A-F | E |

**přechodový graf**

viz obrázek z 4. stránky

**popis pomocných stavových proměnných (pokud je automat používá)**

V naši aplikaci používáme 2 stavové proměnné. První proměnná newState slouží pro uložení stavu, do kterého se program dostane po aplikaci některého z validních symbolů. Toho využijeme hlavně při výběrů svg obrázků pro zobrazení všech možných cest (hran), kam se v nedeterministickém konečném automatu můžeme dostat. Pokud ale zadávaný symbol uživatelem je nevalidní, tak tato proměnná nabije hodnoty -1 pro indikaci konce programu. Druhá proměnná oldState obsahuje indikátor stavu, ve kterém se právě nacházíme. Obě tyto proměnné spolu úzce souvisí, při převodu oldState na newState. Zajišťuje metoda searchInput.

# Implementace kódu třídy NKA

Program je postaven na promítání SVG obrázků na základě vstupu uživatele a mění tyto obrázky reprezentující možné stavy a možné kroky, kterými těchto stavů můžeme dosáhnout symbolem, který uživatel zadal na vstupu. Třída obsahuje několik statických atributů a metod, které spolupracují na zajištění této funkcionality. Zde je podrobný popis jednotlivých částí implementace:

## Atributy třídy

1. **Seznamy znaků**:
    - list0_9, list1_9, list0_7, list0_F, list0, listX – různé seznamy znaků, které představují povolené vstupy pro různé stavy.
2. **SVG canvas**:
    - svgCanvas – objekt třídy JSVGCanvas z knihovny Apache Batik, který umožňuje manipulaci a zobrazování SVG souborů.
3. **Konstanty**:
    - END – znak K označující konec programu.
    - RESET – znak R pro restartování programu.
    - Můžou být napsány i malými písmeny.
4. **Další atributy**:
    - picture – jméno aktuálně zobrazovaného obrázku.
    - newState – proměnná uchovávající následující stav automatu.
    - table – HashMap, klíčem je stav a hodnotou jsou stavy, do kterých se můžeme z klíče dostat.
    - array – dvourozměrné pole seznamů symbolů pro jednotlivé stavy.
    - outputTextField – text pro zobrazení postupně zadávaného řetězce. Vyobrazuje se v textovém poli v dolní části obrazovky na pozici South.
    - textField – textové pole pro zobrazení výstupu.

## Metoda main

1. **Inicializace seznamů znaků**:
    - V této metodě naplníme platnými symboly listy určené pro uchovávání symbolů
2. **Naplňování pole array a tabulky table**:
    - Voláme metody fillArray a fillTable, které naplňují pole array seznamy znaků pro různé stavy a tabulku table definující přechody mezi stavy.
3. **Vytvoření panelu a rozestavení jednotlivých komponent do něj**
    - Využíváme Jframe panel na který vykreslujeme svg obrázky. Dále jsme do něj přidali textField pro zobrazení postupně zadávaného řetězce.

4. **KeyListener pro vstupy uživatele**:
    - Dále metoda main obsahuje reakce na systémové události, kdy pomocí KeyListeneru tyto události obsluhujeme pro zpracování klávesových vstupů uživatele. Konkrétně v metodě keyTyped pomocí metody changeState měníme jednotlivé svg obrázky.

## Zbylé metody

1. **fillArray:**
    - Naplňuje pole array seznamy znaků pro jednotlivé stavy deterministického automatu.
2. **pictureSetter:**
    - Nastavuje název obrázku a aktualizuje zobrazený SVG soubor voláním metody updateSVG.
3. **updateSVG:**
    - Aktualizuje SVG zobrazený v canvasu na základě aktuálního názvu obrázku.
4. **disableTextFieldIfTextFits:**
    - Zakáže focusování textového pole, pokud se zadaný text vejde do jeho šířky. A zároveň umožňuje manipulovat s textem, pokud je delší než šířka textFieldu, za účelem zobrazení celého řetězce.
5. **changeState:**
    - Nejdříve otestuje, jestli předaná stisknutá klávesa uživatelem slouží k ukončení nebo restartování automatu. Následně se aktualizuje proměnná newState (pomocí metody searchInput). Podle obsahu této stavové proměnné se určí, jaký konkrétní svg obrázek se má na panelu vykreslit (pro přesnější určení se využívá i stavová proměnná oldState). Pokud je však newState roven -1, tak ukončí činnost aplikace a vypíše informující hlášku.
6. **searchInput:**
    - Postupně projdeme stavy navazující na stav oldState, které jsme získali z table (hashMapy) a prohledáme listy příslušející k těmto stavům, jestli neobsahují klávesu zadanou uživatelem. Pokud ano, tak vrátíme identifikátor příslušného stavu, nebo -1 v případě, že tento symbol neobsahuje žádný stav, což naznačuje nepovolený znak.
7. **fillTable:**
    - Tato metoda naplní hashMapu, kdy klíč reprezentuje stav, z kterého se dostaneme do stavů uvedených ve values. Tato hashMapa je důležitou konstrukcí, kterou využíváme pro kontrolu, zdali je vstupní symbol validní, neboli nachází-li se v listech spojenými se stavy ve values. Díky použití hashMapy dochází k významné redukci a zpřehlednění kódu, také se tím významně sníží cyklometrická složitost.

# Uživatelská příručka

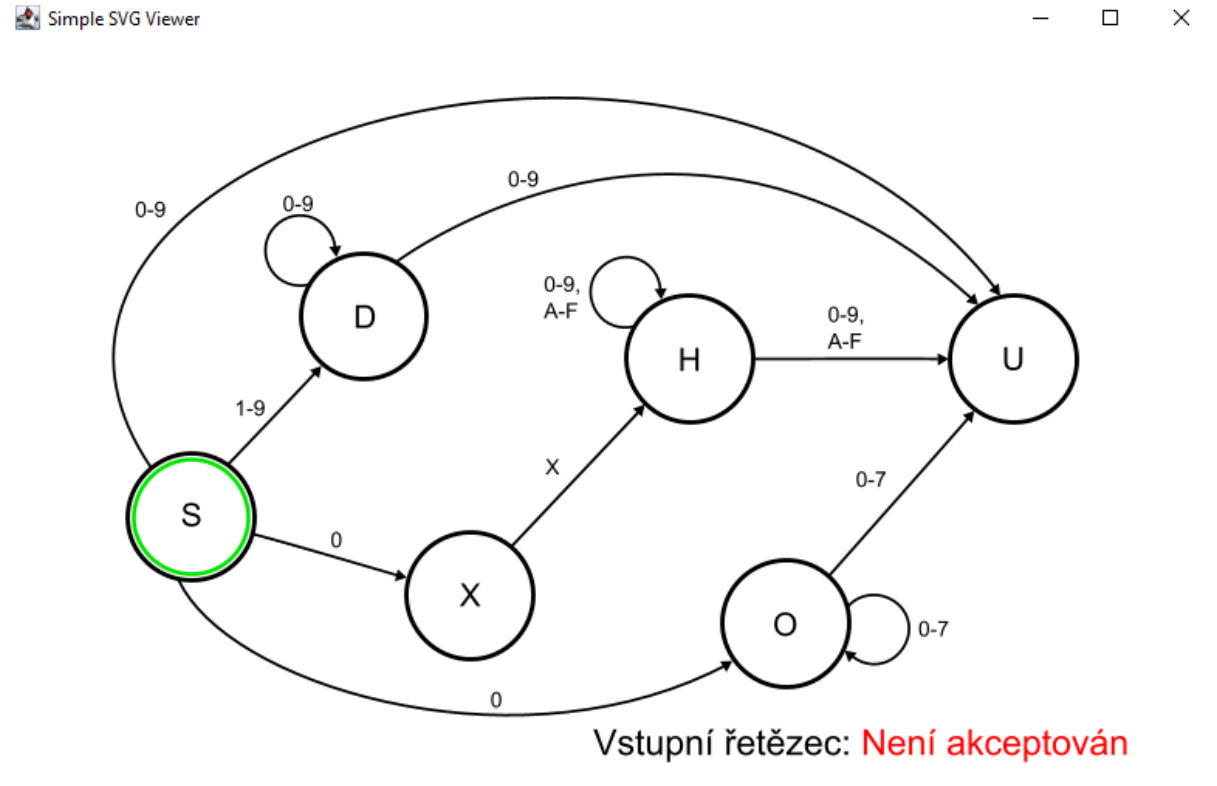
## Spuštění programu

Program lze spustit více způsoby. Prvním způsobem, který je nezávislý na operačním systému, je za pomocí příkazové řádky, a to tak, že aplikace je poskytnuta jako .jar soubor, tudíž uživateli stačí, aby si do systému nainstaloval Java Runtime Environment. Ten Obsahuje Java Virtual Machine (JVM), knihovnu tříd Java Core API a další soubory potřebné pro běh programů v Javě. Pak již pouze stačí ověřit, že v počítači je opravdu JRE nainstalované, a dalším krokem je danou aplikaci spustit například přes příkazovou řádku, a to tak, že otevřeme cmd v adresáři, ve kterém se daná aplikace nachází, a tam zadáme příkaz:

```
C:\Users\Václav\IdeaProjects\Non-determenistic-automat>java -jar NKA.jar
```

*Spouštění .jar přes CMD 1*

Po stisknutí klávesy enter by se měla zobrazit aplikace:

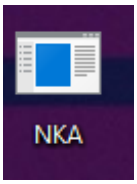



*Vzhled aplikace 1*

Pokud se tak nestalo, buď se nachází problém v adresáři, či staženém JRE.

Druhým způsobem, jak program spustit, je zapnutí aplikace pomocí poskytnutého .exe souboru, který jsme vytvořili za pomocí aplikace launch4J. Ta umožňuje výsledný .jar soubor převést na soubor typu .exe a ten již lze pohodlně spustit, jako každou normální aplikaci, pouze přes zástupce například na ploše. Je důležité upozornit, že tento přístup je možný pouze na

operačním systému Windows, tudíž na Linuxu/MacOS je potřeba aplikaci spouštět pomocí prvního ukázaného řešení, či použití IDE. Nyní si ukážeme způsob spuštění přes .exe soubor:




*.exe soubor 1*

Na ploše bychom měli vidět .exe soubor, ten stačí pouze dvakrát rozkliknout, po rozkliknutí opět můžeme vidět již spuštěnou aplikaci. Vidíme, že tento přístup je opravdu pohodlnější. Pokud tento způsob nefunguje, ověřte kompatibilitu vašeho systému s executable typem souborů.

Je důležité zmínit, že pokud spustíme aplikaci tímto způsobem, a v průběhu běhu aplikace zadáme chybný znak, program se ukončí a neinformuje nás o tom, že ukončení nastalo právě z tohoto důvodu, u spuštění z cmd by příkazový řádek důvod ukončení programu oznámil.

## Komunikace s programem

Program odposlouchává uživatelem zadané klávesy, tedy není potřeba žádného terminálu, stačí pouze zadávat znaky z klávesnice a daná aplikace na ně reaguje. Na obrázku vidíme stavy a k nim odpovídající hrany, u daných hran vidíme symboly, které daná hrana akceptuje, pokud z klávesnice zadáme vstup, který nějaká z hran vedoucí z nynějšího stavu obsahuje, pak se v programu pokračuje přes právě tyto hrany.

Jak můžeme z obrázku vidět, v určitých uzlech automat rozpoznává určité znaky, je potřeba mít na paměti, že symboly typu A-F a X jsou rozpoznávány pouze v případě, jsou-li napsány velkým písmem, to znamená, že pokud tento znak napíšu malým písmem, automat ho nerozpozná a program končí, výjimkou jsou speciální znaky.

Zeleně je v programu označen stav, ve kterém se program momentálně může nacházet, a oranžově je označená hrana, přes kterou se program do daného stavu dostal. S je v programu počátečním stavem a stav U je stavem koncovým, můžeme vidět, zdali byl vstupní řetězec akceptován či nikoliv, tedy zdali alespoň jeden z možných stavů je stavem koncovým.

Program obsahuje takzvané speciální znaky, kterých má program celkem dva. Jsou to znaky RESTART a KONEC, RESTARTU odpovídá na klávesnici znak R, či r, v tom je rozdíl oproti symbolům v aplikaci, v tomto případě je ignorována velikost zapsání znaku. RESTART slouží k restartování programu, tedy program se vrátí do počátečního stavu S a zadaný vstupní řetězec se vymaže, na druhou stranu KONEC je spuštěn na klávesnici za pomocí klávesy K, tím se program celkové ukončí a pro jeho opětovné zapnutí je opět potřeba ho spustit již zmíněnými způsoby.

Pod obrázkem se nachází zadaný řetězec, do tohoto pole není programem umožněno klikat ani psát, a to do té doby, dokud není řetězec delší než šířka pole pro tento řetězec, pak je pro účely zobrazení vstupního řetězce umožněno do pole přistupovat i zobrazovat si požadovanou část zadaného řetězce.


*Ukázka textového pole 1*

## Závěr

Program by se mohl nadále zlepšovat, myslíme si, že například kód by mohl být více členěný, napsaný více objektově orientovaným způsobem, díky tomu by obsahoval i méně statických proměnných.

Dalším problémem, jak jsme již popisovali dříve, je ukončení programu speciálním znakem KONEC, při spuštění aplikace pomocí executable souboru, či ukončení způsobené zadáním nevalidního symbolu, program se tudíž vypne, ale bez toho, aniž by vyhodil hlášku informující, proč byl program vypnut, spuštění prostřednictvím cmd tento problém vyřeší.

Posledním problémem je způsob spouštění aplikace u operačních systému Linux/MacOS, kde jediným přístupem je spuštění skrze příkazovou řádku, což je pracné a nepříliš pohodlné, zlepšení aplikace by tedy tkvělo v přidání dalších přístupů spuštění v těchto operačních systémech.

Myslíme si, že se nám povedlo zobrazení daných přechodů pomocí obrázků, které jsme tvořili pomocí aplikace Inkscape, jelikož jsme chtěli aby obrázky byly vektorové a neztrácely kvalitu při jejich zvětšení, věříme, že nápad implementovat na sebe navazující stavy pomocí hash mapy byl správný přístup, usnadňující práci se zbytkem kódu a ulehčení následného hledání následujících stavů. V týmu se nám pracovalo dobře, jelikož už spolu máme nějakou spolupráci za sebou, a tušíme, v čem je jeden či druhý lepší, a jak toho využít.