Dokumentace úlohy DKA: Determinizace konečného automatu v Python 3 do IPP 2013/2014
Jméno a příjmení: Radek Pištělák
Login: xpiste04

1 Úvod

Úkolem bylo vytvořit skript, který načte konečný automat v jeho textové formě, dle specifikace zadání a případně generuje ekvivalentní deterministický konečný automat bez nedostupných stavů, dle algoritmu z předmětu IFJ.

2 Řešení

Celý projekt je rozdělen do tří souborů. Hlavním souborem je `dka.py`, který nejprve za pomoci třídy `Parser` ze souboru `argparser.py` provede zpracování parametrů. Poté načte vstupní řetězec, který za pomoci regulárních výrazů převede na pětici množin, která je vhodným vstupem pro metodu `create` ze třídy `KA` (soubor `fsm.py`).

Pak jsou dle parametrů volány metody třídy `KA`. Pro pouhé odstranění epsilon přechodů se použije metoda `remove_epsilon_transitions()`, která vrací ekvivalentní konečný automat bez epsilon přechodů. Pokud chceme provést determinizaci je volána navíc i metoda `determinization()`, která vrátí ekvivalentní deterministický konečný automat.

Následuje tisk ve tvaru, dle zadání.

2.1 Zpracování vstupu

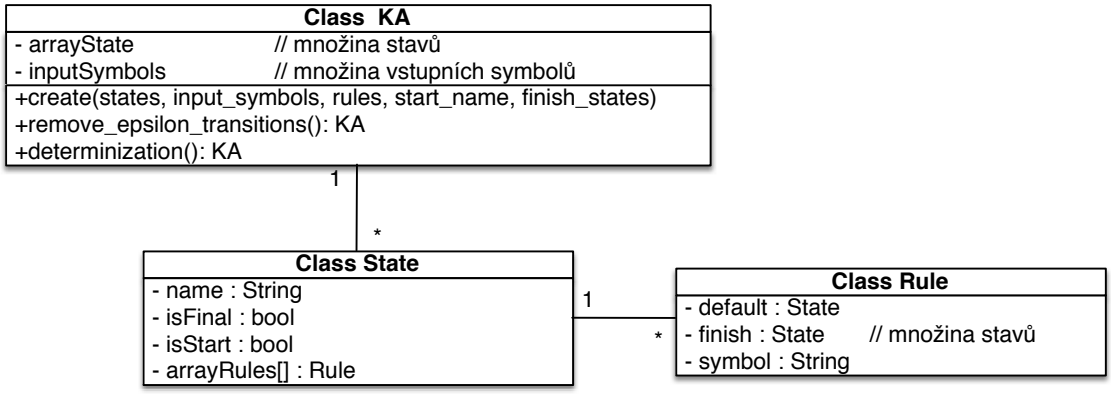
Část zpracovávající vstup začíná regulárním výrazem pro odstraněné komentářů, následuje regulární výraz pro kontrolu a rozdělení vstupu na pětici řetězců, reprezentující množiny dle definice konečného automatu.

V následující části podkapitoly je demonstrován postup zpracování jednotlivých množin na množině stavů, který se principem téměř neliší od ostatních množin.

Nejprve se ze řetězce pomocí několika reg. výrazů odstraní přebytečné mezery. Následně je volána funkce `validator`, které předáme samotný řetězec a regulární výraz popisující stav rozšířený o možnost opakování více stavů za sebou. Funkce poté aplikuje na řetězec reg. výraz a pokud je nalezena shoda nad celým řetězcem vrátí `True` a zbývá už jen rozdělit, řetězec na množinu stavů. Pro rozdělení se využívá metoda `split()` [1]a výsledek je poté uložen do množiny (`set()`).

2.2 Třídy pro práci s konečným automatem

Pro práci s načtenými množinami jsou implementovány třídy `KA`, `State` a `Rule` viz. následující diagram (diagram je velmi zjednodušený a slouží pouze pro ilustraci vztahů).



Obrázek 1: Návrh tříd

[1]V případě dělení řetezců vstupní abecedy a pravidel se používá reg. výraz kvůli případnému výskytu čárky mezi symboly vstupní abecedy.

2.3 Odstranění epsilon přechodů a determinizace

K odstranění epsilon přechodů se využívá metoda `remove_epsilon_transitions()`, která je volána nad třídou `KA` a vrací objekt typu `KA` a je implementována na základě algoritmu z přednášek k předmětu IFJ.

K převedení nedeterministického automatu na deterministický se využívá metoda `determinization()`, která je volána nad třídou `KA` a vrací objekt typu `KA` a je také implementována na základě algoritmu z přednášek k předmětu IFJ.

Pro jejich implementaci je mimo jiné využita třída `MySet`, která je implementována nad třídou `list` ze standardní knihovny a upravuje ji na seznam, který zachovává pořadí, dle vložení a neuchovává duplicity.

3 Rozšíření

3.1 STR

Pokud je voláno rozšíření je nejprve provede determinizace konečného automatu a poté je volána metoda `str()` třídy `KA`, která vrací `True` v případě kladného výsledku analýzy řetězce.

Samotná implementace není složitá. Prochází se řetězec znak po znaku a v každém kroku se zjistí zda se lze přes daný znak dostat do dalšího stavu. Pokud ne metoda vrátí `False` v opačném případě pokračuje v tomtéž pro další znak dokud nenarazí na konec řetězce. Pak se jen zkontroluje jestli je poslední stav, stavem koncovým.

---

4 Závěr autora

Z obávaného zadání se vyklubal jeden z nejzábavnějších projektů, který mě toho i dost naučil ...