Dokumentace úlohy MKA: Minimalizace konečného automatu v Python 3 do IPP 2016/2017
Jméno a příjmení: Pavel Niahodkin
Login: xniaho00

1. Specifikace zadání

Cílem dané úlohy bylo vytvořit script v jazyce Python 3, který bude zpracovávat a případně provádět minimalizaci konečného automatu. Vstupem je pětice komponentů uzavřených do kulatých závorek a oddělených čárkou. První definovaná komponenta je konečná množina stavu konečného automatu. Dále následuje neprázdná vstupní abeceda. Třetí komponentou je definice množiny pravidel. Dále určení počátečního stavu a nakonec poslední komponentou je množina koncových stavů. Nakonec musíme dobře specifikovat podmínky konečného automatu načítaného ze vstupu, pro jeho následnou minimalizaci.

2. Práce se soubory

Pro operace čtení a zápisu do souboru (případně standardního vstupu a výstupu) se používají standardní funkce file.read() a file.write(). Soubor se otevírá pomocí standardní funkce file = open() se zadaným parametrem encoding='utf-8'.

3. Zpracování argumentů

Jednou z nejdůležitějších části řešení úlohy bylo zpracování argumentů. Za jejich zpracování je zodpovědná funkce *get_options()*, která pro svoji práci používá funkci *getopt()* ze standardní knihovny Python. Pokud funkce skončí bez chyb, výsledek je slovník, ve kterém jsou klíče a hodnoty zapsané jako řetězec reprezentující vstupní argument a jeho hodnotu. Jestliže nastane chyba, funkce skončí program s odpovídajícím návratovým kódem. Volání a zpracování výjimky je provedeno uvnitř funkce.

4. Modul finite_automata

Byl vytvořen pro usnadnění práce s konečným automatem, ve kterém jsou definovány následující třídy.

4.1 Třída Rule

Konstruktor dané třídy má tři parametry, které představují současný stav (*state*), vstupní symbol (*input_symbol*) a následující stav (*next_state*). Všechny tyto parametry jsou ve formě řetězců. Takže každý objekt bude mít tři odpovídající atributy. Aby bylo možné dodržet pravidla v Python množině (množina je chápána jako standardní kolekce v Python) bylo nutné redefinovat metody **__hash__** a **__eq__**.

4.2 Třída FiniteAutomata

Konstruktor dané třídy příjmá představu konečného automatu ve formě řetězce, kterou dostáváme podle zadání, jako jediný argument. Syntaktická analýza se provádí pomocí regulárních výrazů z module **re**. Každý atribut objektu odpovídá komponentě z definice konečného automatu. Například, množině stavů odpovídá atribut *self.states* (který je opět chápán jako množina v Python), počátečnímu stavu odpovídá atribut *self.start_symbol*.
V případě semantické nebo syntaktické chyby budou vyvolané výjimky **FASemanticException** a popřípadě **FASyntaxException**. Pro usnadnění zapracování chyb jsou tyhle dvě třídy potomky třídy **FAException**. Objekt vyvolané výjimky v sobě nese kód odpovídající chyby.

4.3 Třída WellSpecifiedFA

Tato třída je potomkem třídy **FiniteAutomata**. Její Konstruktor má jediný parametr, který je objektem třídy **FiniteAutomata**. V konstruktoru této třídy se provádí kontrola, zda je konečný automat dobře specifikovaný. Jestli při kontrole výsledku automatu nebude výsledek obsahovat vlastnosti dobře specifikovaného konečného automatu, bude vygenerována výjimka třídy **FAException** s odpovídajícím kódem chyby. V této třídě se nachází metoda **minimize**, ve které se provádí minimalizace konečného automatu v souladu s algoritmem popsaným v přednášce předmětu IFJ. Implementaci daného algoritmu usnadnilo to, že atributy odpovídají komponentám formální definice konečného automatu.

Závěr a testování

Skript byl vytvořen a dále testován na neočekávané stavy.V průběhu řešení úlohy byl skript testován pomocí testů přiložených k zadání projektu a porovnáván s jejich výsledky. Testování proběhlo na operačními systému Fedora 25. Všechny tyto testy dopadli úspěšně.