DOKUMENTACE ÚLOHY MKA: *Minimalizace konečného automatu v Pythonu do IPP 2015/2016*
JMÉNO A PŘÍJMENÍ: *Adam Bezák*
LOGIN: *xbezak01*

1 Zadání

Naším úkolem bylo naprogramovat skript v jazyce `Python`, který zpracuje a případně minimalizuje konečný automat. Skript zpracovává textový zápis zadaného konečného automatu, kontroluje, zda je dobře specifikován a generuje ekvivalentní minimální konečný automat.

2 Implementace

2.1 Zpracování argumentů

Na zpracování argumentů příkazového řádku jsem využil modulu `argparse`. Zpracování probíhá ve funkci `get_args()`. Za zmínku stojí kontrola duplicity argumentů, protože tuto možnost modul `argparse` nabízí až ve verzi > *3.5*. Řeší se to obyčejnou kontrolou proměnné `sys_argv`, která obsahuje všechny zadané argumenty.

2.2 Zpracování chyb

Výpis a zpracování chyb je řešeno výjimkami. Pokud nastala chyba, vyvolá se výjimka, která se v následujícím bloku odchytí a zpracuje. Výjimka se každým blokem dědí, tak je jednoduché najít i daný řádek kódu, který výjimku vyvolal.

2.3 Vstupní a výstupní soubor

Korektní otevření výstupního, popř. vytvoření výstupního souboru zajišťuje funkce `open_file()`. V případě neúspěchu se vytvoří výjimka, která se odchytí, zpracuje a ukončí program s daným návratovým kódem.

2.4 Kontrola správnosti vstupu

Vstup ze zadaného souboru se nachází v proměnné `input_file_text`. Pokud byl zadán přepínač *-i* nebo *--case-insensitive* tak se za pomoci funkce `lower()` převedou všechna velká písmena na malá. Dále se využívá funkce `remove_comments()`, která odstraní všechny komentáře a nové řádky. V této funkci se také ověří, zda zadaný konečný automat začíná levou kulatou závorkou a končí pravou kulatou závorkou. Samotná podrobná lexikální, syntaktická a sémantická analýza se provádí ve funkci `parse_authomat()`. Ve funkci se vytvoří instance `FSM` třídy `StateMachine`, která reprezentuje konečný automat a ověřuje jeho správnost. Díky faktu, že každá komponenta (kromě počátečního stavu) je ve složených závorkách a od ostatních komponent oddělená čárkou, jsem zvolil konečný automat, avšak který se nekontroluje znak po znaku ale postupně ořezává vstupní řetězec. Každá komponenta se kontroluje stylem oříznutí od začátku znaku `{` po znak `}`. Oříznutý řetězec zavolá příslušnou metodu objektu na kontrolu lexikálních, syntaktických pravidel, případně sématických pravidel. V metodě `get_rules()`, která kontroluje správnost zadaných pravidel, se už pro jednoduchost kontroluje přítomnost epsilon pravidla a nedeterminismus. V případě správně zadaného konečného automatu obsahují atributy objektu dané komponenty konečného automatu.

2.5 Kontrola DSKA

Pokročilá kontrola dobře specifikovaného konečného automatu probíhá zavoláním metody `is_DSKA()` objektu FSM. Cyklem `while`, kterým se přechází od počátečního stavu přes všechny dostupné stavy se kontroluje přítomnost nedostupného stavu. Podobným přístupem, avšak s využitím několika `for` cyklů, se kontroluje počet neukončujících stavů. Pokud průnik prošlých pravidel pravých stran s koncovými pravidly je nulový, tak se zvýší počítadlo neukončujících stavů. DSKA může mít 0-1 stavů které jsou neukončující.

2.6 Minimalizace

V případě minimalizace se volá metoda objektu `FSM_Minimize()`. Do seznamu `all_states` se uloží množina koncových stavů a jejich doplněk k ostatním stavům. Vybere se množina stavů a s každým prvkem vstupní abecedy se zjišťují pravé strany pravidel. Pokud pravé strany pravidel nepocházejí z jedné množiny stavů, nastává štěpení stavů. Podmínkou

je, že průnik pravých stran s množinami v seznamu `all_states` musí být větší než 0 a menší než délka seznamu pravých stran. Dále je třeba najít ke každé pravé straně i odpovídající levou stranu pravidla. Ze seznamu `all_states` je třeba odstranit štěpenou množinu a přidat novou podmnožinu levých stran. Po algoritmu minimalizace je potřeba spojit nové stavy a vytvořit odpovídající pravidla, koncové stavy a počáteční stav. Tvorba těchto stavů je zajištěna integrovanými funkcemi jako `join()` a `Sorted()`.