Dokumentace úlohy MKV: Minimalizace konečného automatu v Python 3 do IPP 2016/2017
Jméno a příjmení: Marek Šipoš
Login: xsipos03

1 Analýza požadavků

Zadáním úlohy byla tvorba skriptu v jazyce Python, který by umožnil načíst vstupní soubor[1] obsahující textový zápis stavového automatu v určitém formátu, tento automat poté zkontroloval pro formální správnost a provedl nad ním určité operace definované přepínači. Nakonec bylo potřeba výsledný stavový automat normalizovaně vypsat.

2 Implementace

Následující část popisuje samotný způsob implementace jednotlivých částí projektu, od parametrů, přes práci se soubory a další dílčí části úlohy.

2.1 Načítání parametrů

Skript měl umožňovat korektně načíst určitou sadu parametrů a také na ně reagovat. Parametry měly umožňovat i zkrácený zápis s jedním spojovníkem. Pro řešení tohoto problému byla vybrána knihovna jazyka Python s názvem `argparse`, která tento problém pozoruhodně jednoduše řeší. Stačí vytvořit instanci určité třídy, metodami nad touto instancí určit parametry a ty jsou po načtení dostupné ve slovníku (*dictionary*) dané instance.

Problém nastal při zpracování možných chybových stavů (jako například zadání parametru s hodnotou, ačkoliv daný parametr hodnotu nevyžaduje), které byly doprovázeny špatnými chybovými kódy neodpovídajícími zadání a navíc s nechtěným výstupem navíc. Tento problém byl vyřešen děděním od původní třídy `ArgumentParser` a přepsáním její metody error, která toto chování definuje.

2.2 Práce se soubory

Jakékoliv operace vstupu a výstupu obsahovaly podmínky na výskyt parametrů `input` a `output` a podle toho došlo k rozhodnutí, zda bude využito standardního vstupu (resp. výstupu), nebo souborů. Pokud došlo k nějaké chybě (např. nenalezení souboru ke čtení nebo nedostatečná oprávnění k souboru pro zápis), byly tyto chyby (výjimky) podchyceny konstrukcí `try-except` a bylo na ně patřičně reagováno.

2.3 Validace a načítání stavového automatu

Problematika validace a načítání stavového automatu se ukázala býti poměrně dosti komplikovanou, proto bylo využito patřičných technik, které budou dále rozvedeny.

2.3.1 Objektový přístup

Pro práci se stavovým automatem a jeho složkami byly vytvořeny třídy `FSM, State, Symbol` a `Rule`, které v sobě uchovávaly vnitřní informace (např. identifikátory). Tento přístup se později ukázal jako velice výhodný, protože **značně** zlepšil přehlednost a kvalitu kódu, jakožto i umožnil některé programovací techniky (např. předávání referencí v rekurzivní funkci). Třídní model byl využit také pro definici výčtových listů.

2.3.2 Lexikální analýza

Pro jednoduchou identifikaci jednotlivých částí vstupu bylo využito lexikální analýzy. V rámci lexikální analýzy byla vytvořena třída `Token`, která si uchovávala informaci o jejím typu (reprezentovaným výčtovým listem `Lexem`) a vnitřní data (identifikátor) v textové podobě. Instance této třídy pak byly generovány funkcí `get_next_token`, která za pomocí stavového automatu procházela vstup, identifikovala jej a poté jej rozdělila na již zmíněné Tokeny. Ty pak byly dále využity v syntaktickém analyzátoru.

[1] nebo přímo standardní vstup

2.3.3 Syntaktická analýza

Pro ověření syntaktické správnosti zadaného stavového automatu byla využita **prediktivní syntaktická analýza** (shora dolů), která využívala LL-tabulky a zásobníku obvyklým způsobem. Následují použitá gramatická pravidla (nebo též *pravidla derivací*):

```
1  FSM => ( STATES , INPUTS , RULES , state , STATES )
2  STATES => { FIRST_STATE }
3  FIRST_STATE => state ANOTHER_STATE
4  FIRST_STATE => eps
5  ANOTHER_STATE => , state ANOTHER_STATE
6  ANOTHER_STATE => eps
7  INPUTS => { FIRST_INPUT }
8  FIRST_INPUT => input ANOTHER_INPUT
9  FIRST_INPUT => eps
10 ANOTHER_INPUT => , input ANOTHER_INPUT
11 ANOTHER_INPUT => eps
12 RULES => { FIRST_RULE }
13 FIRST_RULE => state input -> state ANOTHER_RULE
14 FIRST_RULE => eps
15 ANOTHER_RULE => , state input -> state ANOTHER_RULE
16 ANOTHER_RULE => eps
```

Velkými písmeny jsou uvedeny non-terminály, malými terminály. Výraz 'eps' značí epsilon (prázdný řetězec).

2.3.4 Sémantická analýza

Před syntaktickou analýzou proběhla definice pomocných funkcí pro pozdější provedení sémantické analýzy. Jednalo se hlavně o funkce, které zjednodušily zpracování a uložení stavového automatu a jeho průběžnou kontrolu. Samotná sémantická analýza probíhala spolu se syntaktickou analýzou.

Zajímavá byla funkce `rules_build`, kterou je nutné pro vložení jednoho pravidla do stavového automatu volat třikrát. Důvodem, proč tato funkce funguje takto, je ten, že jednotlivé složky pravidla (výchozí stav, čtený symbol a výsledný stav) jsou načítány postupně (tak, jak probíhá syntaktická analýza). Tato funkce si za použití pseudo-statiky tyto složky postupně uchovávala a až jich měla dostatek pro tvorbu pravidla, pravidlo vytvořila a vložila do předaného stavového automatu.

2.3.5 Formální validace

Nakonec byla provedena formální validace zadaného stavového automatu. Ověřoval se hlavně determinismus automatu, jeho úplnost a případné další chyby, jako více než jeden neukončující stav (viz níže) nebo hledání případných nedosažitelných stavů.

2.4 Hledání neukončujících stavů

Hledání neukončujících stavů (pokud to je vyžadováno) se provádí rekurzivním voláním funkce `get_reachable`, která jako parametr požaduje referenci na množinu již iterovaných stavů a stav výchozí. Tato funkce prochází celý graf stavového automatu a nad každým stavem volá sebe sama. Každý výchozí stav se uloží do již zmíněné společné množiny. Nakonec se provede porovnání vytvořené množiny s množinou všech stavů daného stavového automatu. Pokud nějaké stavy chybí, jsou to jistě stavy neukončující. Tato funkce byla využita i pro hledání nedosažitelných stavů.

2.5 Minimalizace

Minimalizace byla provedena standardním způsobem (štěpením na stále menší skupiny). Skupina byla v tomto případě reprezentována třídou `Group`, která obsahovala číselný identifikátor, množinu instancí stavů a speciální textové razítko ve formátu `X|x>n|x>n|...` kde `'X'` reprezentoval výchozí číslo skupiny (protože dva prvky se stejným razítkem lze sloučit pouze tehdy, pocházejí-li ze stejné skupiny), `'x'` reprezentoval čtený znak, `'n'` reprezentoval číslo skupiny a `'>', '|'` byly oddělovače. Toto razítko sloužilo jako jakýsi jednoznačný extrakt z tabulky pravidel, který jasně definoval a snadno identifikoval prvky, které pro stejné vstupní symboly přecházejí do stavů stejné skupiny.

Minimalizace byla provedena tak, že se nejprve vytvořily dvě základní skupiny (koncové stavy a ty ostatní) a jejich stavy se prošly. Pro každý stav se iterovalo seřazenou abecedou a vytvořilo se již zmíněné jednoznačné razítko, na základě kterého byl prvek zařazen do skupiny nové. Jakmile byly všechny prvky zařazeny podle svých razítek do jednotlivých skupin, proces se opakoval, dokud se počet skupin nepřestal měnit.

Nakonec bylo potřeba si ze skupin vytvořit nové (sloučené) stavy nazvané podle zadání s podtržítkem jako oddělovačem a převést i ostatní části z pětice definující konečný automat tak, aby odpovídaly nově vzniklým stavům.