Dokumentace úlohy: MKA: Minimalizace konečného automatu v Pythonu do IPP 2014/2015
Jméno a příjmení: Jan Kubiš
Login: xkubis13

Dokumentace k 2. projektu IPP 14/15

(MKA: Minimalizace konečného automatu)

- **Zadání úlohy**

  Skript „mka.py“ je implementován v jazyce Python. Slouží pro práci s konečným automatem, především pro jeho minimalizaci. Vstup i výstup jsou v textové formě. Odkud se čte a kam se zapisuje volí uživatel parametry při spouštění.

- **Argumenty příkazové řádky a jejich zpracování**

  Skript lze spustit jak bez argumentů, tak s některými z následujících:

  1. *--input*=PATH určuje, kde odkud má skript číst zadaný konečný automat. Při vynechání tohoto parametru se autot čte ze standartního vstupu. Při jeho použití musí být doplněn o validní cestu PATH k souboru. Je-li zadána nevalidní, skript končí s návratovou hodnotou 2.
  2. *--output*=PATH určuje, kam se bude zapisovat veškerý výstup skriptu. Není-li uveden, proběhne výpis na standartní výstup. Pokud je zadán s nevalidní cestou PATH, skript je ukončen a vrací hodnotu 3.
  3. *-m* zadává se, chceme li automat na vsutpu minimalizovat. Nelze kombinovat s parametrem *-f*.
  4. *-f* automat na vstupu je zpracován, obsahuje-li právě jeden neukončující stav, je tento vypsán jako výstup skriptu. Nelze kombinovat s parametrem *-m*.
  5. *-i* při načítání automatu se nehledí na velikost písma.

  Nápověda se zobrazí při spuštění s parametrem *–help*.Pokud nebyl zadán přepínač *-m* ani *-f*, na výstupu je vypsán automat, který byl zadaný – avšak v normalizované formě. Po zkušenostech z předchozího projektu jsem si na argumenty udělal rovnou vlastní parser. Zkontroluje validitu a nastaví řídící proměnné skriptu.

- **Zpracování vstupu**

  Ze všeho nejdřív je konečný automat na vstupu prohnán naimplementovaným konečným automatem, který z něj odstraní všechny komentáře, a taky nahradí řídící znaky pro popis automatu (obyčejné závorky, složené závorky, čárka, apostrof, znak začátku komentáře) vlastním textem – ty se totiž mohou objevit i jako symbol vstupní abecedy, a proto je potřeba je rozlišit. Řešit tento problém regulárními výrazy mi přišlo zdlouhavé. Stačilo zavést automat, který se řídí pomocí boolean proměnné držící informaci o tom, zda-li se nacházíme ve stavu zapisování symbolu vstupní abecedy nebo mimo něj. To je asi vše co se dá v této chvíli dělat – následuje kontrola lexikální a syntaktické validity zápisu automatu pomocí regulárních výrazů. Například odstranění případných mezer není nyní možné (*Q={s1,s␣2,s3}* je chybný zápis, avšak *Q={s1,s2,s3}* už správný). Celek je reguláry rozdělen a každá komponenta pětice nahrána zvlášť. V dalším kroce probíhá jejich kontrola. Položky komponenty jsou nahrány do seznamu jejím rozdělením v místě řídící čárky. Vypuštění všech bílých znaků může proběhnout po kontrole jednotlivých položek – regulární výrazy provádějící tuto kontrolu tedy musí být doplněny o možné bílé znaky na správných místech. Prázdný řetězec není symbol, pokud se tedy objeví ve vstupní abecedě, chyba se hlásí odtud. Takto probíhá **lexikální a syntaktická kontrola** zadaného automatu.

Po dokončení předchozí části proběhne **sémantická kontrola**. Položky všech komponent jsou uloženy v poli v normalizovaném tvaru, můžou se tedy vzájemně porovnávat. Konkrétně počáteční stav, koncové stavy a části položek přechodových pravidel jsou kontrolovány, zda-li neobsahují stav resp. symbol který nepatří do množiny všech stavů resp. do vstupní abecedy. Pokud ano, skript hlásí sémantickou chybu. Stejnětak je-li množina reprezentující vstupní abecedu prázdná.

Po dokončení předchozí části proběhne **kontrola na dobrou specifikovanost** automatu. Je vyžadováno, aby pro každý stav existovalo právě jedno (determinismus) přechodové pravidlo pro každý symbol. Kontroluje se tedy počet pravidel se součinem kardinalit zmíněných množin a také shodnost levých stran pravidel. Je-li v pravidlech uveden epsilon přechod, chyba se hlásí odtud. Nakonec je potřeba testovat existenci nedostupných a neukončujících stavů (při nálezu nedostupného stavu je hlášena chyba, neukončující stav může být maximálně jeden – ten se zde i případně nahraje do speciální proměnné, se kterou se pracuje, byl-li zadán argument *-f*).

- **Tvorba výstupu**

V této chvíli máme načtený dobře specifikovaný automiat. Výstup záleží na zvoleném přepínači *-m* /*-f*. Pokud uživatel nezvolil ani jeden, obsah komponent automatu je lexigofraficky uspořádán, poté podle pravidel normalizovaného výstupu poskládán do výstupního řetězce. Ten je následně vypsán na výstup zvolený argumentem *--output*. Volil-li argument *-m*, ještě před uspořádáním a výpisem je automat minimalizován. V případě argumentu *-f* se pouze vypíše nalezený neukončující stav (případně 0) a skript končí.

- **Minimalizace**

Algoritmus minimalizace z přednášek je na pochopení poměrně jednoduchý. Implementace však už tak jednoduchá nebyla. Štěpení množiny stavů se mi nepodařilo zajistit jinak než spoustou zanořených cyklů – vzhledem k tomu, že se potřebují prostřídat všechny možné kombinace prvků pěti množin to nejspíš ani jinak možné není. Za největší překážku považuju problém rozdělení množin podle stavu na pravé straně vybraných pravidel. V ukázkovém řešení minimalizace v předmětu IFJ je totiž popsán ideální případ, kdy se množiny buď vůbec nedělí a nebo se dělí na dvě další. Při jiném zadání by se ale mohlo stát, že se bude rozdělovat po X prvcích na Y množin. Po dlouhém přemýšlení ale vyšlo najevo, že toto není potřeba řešit – stačilo vždy příslušnou množinu rozdělit ne na Y, ale na 2 množiny. Zbylá množina stavů je totiž rozšštěpena (je-li to možné) hned v dalším průchodu (zajišťuje jej cyklus *while*, který pokračuje, dokud se množina mění – je co rozdělovat). Před dalším průchodem jsou nalezené prvky odebrány z míst v původní množině ke štěpení a jsou přidány jako další samostatná množina. Pokud je štěpení hotové, podle výsledků jsou přetvořeny komponenty KA – množina stavů, pravidel, počáteční stav a množina koncových stavů.