Dokumentace úlohy MKA: Minimalizace konečného automatu do IPP 2011/2012

Jméno a příjmení: Vendula Poncová
Login: xponco00

Zpracování parametrů

Pro zpracování parametrů se volá funkce `processParams`. Zvolila jsem ruční zpracování parametrů, neboť modul `argparse` neumožňoval zakázat zkracování a shlukování parametrů. Všechny povolené tvary parametrů jsem uložila do slovníku jako klíče, kde odpovídající hodnotou je identifikátor parametru. Rozpoznání zadaného parametru, pak probíhá jednoduše dotazem, zda je daný klíč definovaný ve slovníku. Pokud ano, vloží se do slovníku zpracovaných parametrů klíč *identifikátor parametru* s hodnotou *uživatelem zadaná hodnota parametru*. Před vkládáním testuji, zda klíč již neexistuje, případně vyvolám chybu. Parametry s hodnotou a booleovské parametry zpracovávám zvlášť.

Práce se vstupem a výstupem

Abych omezila práci se soubory, načítám ve funkci `readInput` vstup do řetězce, který dále předávám jako parametr funkcím. Funkce opět vrací řetězce, které tisknu na výstup ve funkci `writeOutput`.

Konečný automat

Třída `FSM` reprezentuje konečný automat. Atributy objektu jsou množiny `Q`, `E`, `F`, proměnná `s` a slovník `R`, které odpovídají pětici definující konečný automat. Slovník `R` uchovává pravidla ve tvaru `R` [*stav 'symbol'*] = *stav_po_přechodu*. Přechod mezi stavy je pak jen otázkou vytvoření příslušného klíče a přístupu k hodnotě ve slovníku.

Funkce `prettyprint` vrací řetězec s výpisem normalizové formy automatu. Z datových struktur jsou vytvořeny seznamy a setříděny metodou `sort`. Oddělující čárky mezi prvky komponent řeším metodou `join`.

Kontrolu typu automatu provádí funkce `checkDKA` a `checkDSKA`, které ověřují, zda je konečný automat deterministický konečný automat, respektive dobře specifikovaný konečný automat. Během kontroly na dobře specifikovaný konečný automat se volají funkce `findNonAvailable` a `findNonFinishing`, které vrací množinu nedostupných, resp. neukončujících stavů. Tyto funkce dále využívám v rozšíření MWS. Hledání neukončujících stavů jsem implementovala podle algoritmu uvedeného v materiálech předmětu IFJ. Generuji množinu stavů, které jsou ukončující a vracím rozdíl množiny stavů s touto množinou. Hledání dostupných stavů jsem implementovala obdobně. Postupuji od počátečního stavu a generuji množinu dostupných stavů, vracím opět rozdíl množin.

Parser

Načítání konečného automatu ze řetězce obstarává třída `FSM_parser`. Pri inicializaci objektu se provede kompilace regulárních výrazů popisujících strukturu automatu, stav a symbol a uloží se do proměnných objektu. Vlastní parsování se spouští metodou `run`. Ze vstupního řetězce se odstraní komentáře a text se pomocí regulárního výrazu rozdělí na jednotlivé komponenty. Komponenty zpracovává funkce `matchGroup`, která hlídá, zda jsou prvky komponenty oddělené čárkou. Správný tvar prvku testuje funkce `matchItem`, jejímž parametrem je regulární výraz popisující prvek. Pravidla jsou zpracovány zvlášťve funkci `matchRule`, která využívá definované regulární výrazy stavu a symbolu.

Vlastní parsování probíhá tak, že si pamatuji konec poslední úspěšné „match“ operace. Regulární výrazy přijímají všechny bílé znaky před prvkem i za prvkem, proto lze ihned testovat následující znak na oddělovač (čárku). Posunu index konce zpracované části řetězce za oddělovač a pokračuji kontrolou dalšího prvku. Pomocí skupin regulárních výrazů ukládám zpracované položky do struktur objektu `FSM`.

Třída vrací nový objekt konečného automatu, který odpovídá zadaným lexikálním a syntaktickým pravidlům.

Minimalizace

Minimalizace konečného automatu se spouští metodou `run` ve třídě `FSM_minimize`. Vstupem je dobře specifikovaný konečný automat a výstupem ekvivalentní minimalizovaný. Algoritmus minimalizace pracuje se slovníkem `mini_states`, který obsahuje stavy minimalizovaného automatu. Klíčem je identifikátor stavu vygenerovaný podle specifikace, hodnotou množina stavů, které se sloučily v jeden stav. Algoritmus je pak implementovaný dle materiálů předmětu IFJ. Když procházím stavy z množiny sloučených stavů, provedu přechod a hledám identifikátor, popisující množinu stavů, ve které se stav po přechodu vyskytuje. Na základě tohoto identifikátoru rozděluji

stavy do dvou množin a pokud druhá z množin bude po ukončení cyklu neprázdná, provede se štěpení. Ve slovníku `mini_states` se zruší položka štěpeného stavu a přidají se dva nové stavy.

Pokud nedojde k žádné nové změně slovníku `mini_states`, vytvoří se na základě tohoto slovníku nový automat. Počáteční stav a pravidla původního automatu se transformují tak, že se stavy nahradí příslušnými identifikátory sloučených stavů. Do množina koncových stavů se přidají všechny identifikátory, které ve své množině zahrnují alespoň jeden původní koncový stav. Množina stavů se převezme ze slovníku `mini_states`. Abeceda je pro původní a minimalizovaný stav stejná.

Rozšíření

V modulu `FSM` jsem implementovala rozšíření MST a MWS. Funkce `analyzeString` vyhodnocuje, zda automat přijímá zadaný řetězec. Na začátku se inicializuje stav na počáteční. Pro všechny znaky zadaného řetězce se ověří, zda je znak symbolem abecedy a zda pro daný stav a znak existuje pravidlo. Pokud ano, provede se přechod. Po ukončení cyklu se testuje, zda automat skončil v koncovém stavu a funkce vrátí odpovídající hodnotu. Existence pravidla se provádí ověřením existence klíče *stav* '*znak*' ve slovníku `R`, přechodem je myšleno nastavení proměnné stavu na hodnotu příslušející klíči.

Funkce `transformToDSKA` provádí transformaci deterministického koncového automatu na dobře specifikovaný konečný automat. Využívám již zmíněné funkce `findNonAvailable` a `findNonFinishing`, které vyhledají nedostupné a neukončující stavy. Tyto stavy pak odstaním z množiny stavů a množiny koncových stavů a vymažu pravidla, kterých jsou součástí. Dalším krokem je doplnění do úplného automatu. Pro všechny stavy procházím celou abecedu a pokud nějaké pravidlo chybí, vygeneruji nové s přechodem do pasti (stav *qFalse*). Pokud jsem byla nucená použít past, doplním pravidla pro *qFalse*. Nakonec zkontroluji, zda jsem ze stavů nevyřadila počáteční stav. Pokud ano, nastavím počáteční stav na *qFalse*.