Zadání úlohy do projektu z předmětu IPP 2016/2017
(Obecné a společné pokyny všech úloh jsou v `proj2017.pdf`)

MKA: Minimalizace konečného automatu

Zodpovědný cvičící: Zbyněk Křivka (krivka@fit.vutbr.cz)

1 Detailní zadání úlohy

Vytvořte skript pro zpracování a případnou minimalizaci konečného automatu. Skript bude zpracovávat textový zápis zadaného konečného automatu, validovat, že je dobře specifikovaný, a generovat ekvivalentní minimální konečný automat přesně podle algoritmu z 11. přednášky (snímek 23/35) předmětu Formální jazyky a překladače (IFJ).

1.1 Formát vstupu

Komentáře do konce řádku[1] začínají znakem `#`. Bílé znaky jako konec řádku (`\n` i `\r`), mezera či tabulátor jsou ignorovány[2] (až na později definované případy).

Stav je reprezentován identifikátorem jazyka C, který nezačíná ani nekončí podtržítkem. Vstupní symbol je reprezentovaný libovolným znakem uzavřeným v apostrofech[3]. U stavů i vstupních symbolů záleží na velikosti písmen[4]. Jako vstupní symboly lze využít i metaznaky popisu automatu a všechny bílé znaky, tj. `'('`, `')'`, `'{'`, `'}'`, `''''`, `'-'`, `'>'`, `','`, `'.'`, `'#'`, `' '`, znak konce řádku v apostrofech a znak tabulátoru v apostrofech. Apostrof musí být navíc uvnitř apostrofů zdvojený. Prázdná dvojice apostrofů `''` reprezentuje *prázdný řetězec*[5].

Celý konečný automat je zapisován podobnou notací jako ve formálních jazycích (IFJ). Celý konečný automat je pětice uzavřená do kulatých závorek. Každá komponenta kromě komponenty určující počáteční stav je dále uzavřena ve složených závorkách a od ostatních komponent oddělena čárkou. Jednotlivé prvky množin reprezentujících komponenty (opět kromě počátečního stavu) jsou odděleny také čárkou.

Nejprve je definována konečná množina stavů, následuje neprázdná vstupní abeceda, poté definice množiny pravidel, dále určení počátečního stavu a nakonec množina koncových stavů. Množina pravidel je popsána seznamem pravidel. Každé pravidlo je zapsáno ve tvaru: $p\,a$`->`q, kde p je *výchozí stav*, a je *čtený vstupní symbol* (případně reprezentace prázdného řetězce), následuje dvojznak pomlčka[6] s větším reprezentující šipku (tento dvojznak nesmí být rozdělen jiným znakem) a poslední část pravidla q určuje *cílový stav*.

Příklad vstupního zápisu konečného automatu:

```
# Příklad konečného automatu ve vstupním formátu úlohy MKA
({s, f,q4,q2 , q1, # nějaký komentář
  q3}, # nějaký komentář
  {'á', ')' }, {
```

[1] Nenásleduje-li za komentářem konec řádku, je komentář validně ukončen koncem souboru.
[2] Nelze však vypuštěním bílého znaku získat ze dvou identifikátorů jeden.
[3] ASCII kód 39 (POZOR! Přes schránku se z PDF tento symbol kopíruje často špatně.)
[4] Tj. definice automatu je case-sensitive.
[5] V IFJ byl prázdný řetězec značen řeckým písmenem ε. V této úloze je použita jiná reprezentace.
[6] Z pohledu české typografie se jedná o tzv. spojovník reprezentovaný ASCII znakem s ordinální hodnotou 45.

```
  s 'á'->f, f  'á' ->s, s')'->q3, s ')' -> q4,
  q1 'á'->q1, q2'á' -> q2, q3 'á'->q4, q4 'á'->
  q3, q1 ')' -> s, q2')'
  ->f , q3 ')'->q1, q4')'->q2  # další komentář
}, # následuje komponenta definující počáteční stav
s
, {f, s } ) # koncové stavy a ukončení definice automatu
# zde může následovat libovolný počet bílých znaků nebo komentářů
```

1.1.1 Kontrola správnosti vstupu

Vstupní automat nesplňující popsaná lexikální a syntaktická pravidla ukončí skript s chybou a návratovým kódem `60`. Sémantická chyba a ukončení skriptu s návratovým kódem `61` nastane v následujících případech:

- vstupní abeceda je prázdná,
- pravidlo obsahuje stav resp. symbol, který není v množině stavů resp. vstupní abecedě,
- počáteční stav nepatří do množiny stavů,
- množina koncových stavů není podmnožinou množiny stavů.

Při opakování stejných pravidel/stavů/symbolů v rámci jedné komponenty jsou tato ve výsledné množině pouze jednou (tj. množina na výstupu není multimnožinou). Při načítání vstupu je také automaticky prováděna kontrola, že se skutečně jedná o dobře specifikovaný konečný automat (viz přednášky IFJ). Pokud vstup nereprezentuje dobře specifikovaný konečný automat, skončí skript s chybou a vrátí návratový kód `62`. Kombinace více chyb nebude testována.

1.2 Transformace a formát výstupu

Popis algoritmu je odkázán v referencích (Kapitola XI, snímek 23/35). Algoritmus je kvůli testování výsledků závazný. Po načtení automatu je třeba provést kontrola, že se jedná skutečně o dobře specifikovaný konečný automat (viz Kapitola IV, snímek 34/36), a pak pomocí algoritmu z Kapitoly XI a snímku 23/35 můžeme provést minimalizaci.

Při minimalizaci se provádí tzv. *slučování a štěpení stavů*. Pro jednotný výsledek bude potom výsledný identifikátor pro štěpený stav definován jako spojení všech původních stavů (resp. jejich identifikátorů) pomocí znaku podtržítko v lexikografickém vzestupném pořadí (pořadí jednotlivých znaků je určeno jejich ordinální hodnotou). Pokud dojde k atomickému štěpení až na původní stav, tak se samozřejmě jeho identifikátor využije jako identifikátor i ve výsledném automatu. Například, pokud dostaneme po minimalizaci stav reprezentující množinu původních stavů {`s1`, `p2`, `p`, `P2`}, tak výsledný stav bude mít identifikátor `P2_p_p2_s1`, kdy jsou jednotlivé stavy spojeny podtržítkem v lexikografickém pořadí. Není třeba uvažovat kolize, které mohou potenciálně vzniknout kvůli slučování stavů obsahujících podtržítka již v původní množině stavů.

1.2.1 Normální forma výstupu

Výstupní formát výsledného konečného automatu vychází ze vstupního formátu a je definován následující normální formou. Všechny komentáře a nadbytečné bílé znaky budou vypuštěny. Automat bude vypsán v úplném tvaru a s každou komponentou začínající na zvláštním řádku. Kromě množiny

pravidel budou všechny komponenty právě na jednom řádku[7]. Za každou komponentou bezprostředně následuje čárka a odřádkování[8]. Za poslední komponentou nebude čárka ale pouze odřádkování, takže uzavírací kulatá závorka bude na novém řádku. Za uzavírací kulatou závorkou bude ještě odřádkování. Stavy v množině stavů, resp. symboly ve vstupní abecedě, budou odděleny čárkou a jednou mezerou (za posledním prvkem nebude čárka ani mezera) a ve svých komponentách seřazeny lexikograficky vzestupně. Každý symbol vstupní abecedy bude opět uveden v apostrofech.

Každé pravidlo z množiny pravidel bude začínat na novém řádku (tj. odřádkování bude i za otevírající levou složenou závorkou; uzavírající pravá složená závorka bude též na novém řádku). Oddělující čárka bude bezprostředně za identifikátorem cílového stavu a za ní rovnou odřádkování. Množina výsledných pravidel bude seřazena vzestupně do seznamu pravidel podle sdruženého klíče ze dvou podklíčů. Primární klíč je identifikátor výchozího stavu (řazeno lexikograficky podle ordinálních hodnot znaků) a sekundární je znak reprezentující vstupní symbol bez případných uvozujících apostrofů (řazeno podle ordinální hodnoty znaku, prázdný řetězec má hodnotu 0).

Vstupní symbol i prázdný řetězec obsažený v pravidlech bude na výstupu vždy uveden v apostrofech (případný apostrof jako vstupní symbol bude zdvojen). Části pravidla budou odděleny právě jednou mezerou, jak je vidět na příkladu formátování jednoho pravidla na výstupu (včetně oddělující čárky za pravidlem):

```
stav1_stav2 'a' -> stav2_stav3,
```

Tento skript bude pracovat s těmito parametry:

- `--help` viz společné zadání všech úloh.
- `--input=`*filename* zadaný vstupní textový soubor *filename* v UTF-8 s popisem dobře specifikovaného konečného automatu.
- `--output=`*filename* textový výstupní soubor *filename* (opět v UTF-8) s popisem výsledného ekvivalentního[9] konečného automatu v předepsaném formátu výstupu.
- `-f`, `--find-non-finishing` hledá neukončující stav zadaného dobře specifikovaného konečného automatu (automat se nevypisuje). Nalezne-li jej, bez odřádkování jej vypíše na výstup; jinak vypíše pouze číslici `0`. (Před hledáním se provede validace na dobrou specifikovanost automatu.) Parametr nelze kombinovat s parametrem `-m` (resp. `--minimize`).
- `-m`, `--minimize` provede minimalizaci dobře specifikovaného konečného automatu (viz algoritmus IFJ, přednáška 11, snímek 23/35). Parametr nelze kombinovat s parametrem `-f` (resp. `--find-non-finishing`).
- `-i`, `--case-insensitive` nebude brán ohled na velikost znaků při porovnávání symbolů či stavů (tj. `a = A`, `ahoj = AhOj` nebo `A_b = a_B`); ve výstupu potom budou všechna velká písmena převedena na malá.

Pokud nebude uveden parametr `-m` ani `-f`, tak dojde pouze k validaci načteného dobře specifikovaného konečného automatu a k jeho normalizovanému výpisu.

[7] Výjimkou je případ, kdy je vstupním symbolem znak konce řádku.

[8] Odřádkování provádějte unixovým způsobem (znakem `LF`).

[9] Ekvivalentní konečný automat je definován jako automat přijímající stejný jazyk.

2 Bonusová rozšíření

- **WHT** (1 bod): Volba `-w`, `--white-char` ve vstupu lze oddělující čárku nahradit libovolným bílým znakem (i komentář je brán jako bílý znak) a dále mohou být vypuštěny apostrofy u prázdného řetězce i vstupních symbolů[10] (až na metaznaky a bílé znaky). Příklad:
```
({s f q4 q2 , q1, q3} {á # nějaký komentář
')' }, { s á->f, f'á'->s s ')'->q3, s')' ->q4 q1'á'->q1 q2 á-> q2 q3 á ->q4
q4'á'-> q3, q1 ')' -> s q2')' ->f q3 ')'->q1, q4')'->q2 }
s {f, s } )
```

- **RLO** (0,5 bodu): Volba `-r`, `--rules-only` vstupní soubor obsahuje zkrácený vstupní zápis tj. pouze množinu pravidel automatu (nikoli ostatní komponenty). Ve zkráceném zápisu jsou na vstupu pouze jednotlivá pravidla (bez složených závorek ohraničujících množinu pravidel v úplném zápisu). U pravidla může navíc bezprostředně za cílovým stavem následovat tečka, která označuje cílový stav zároveň jako koncový (tečku není třeba opakovat u každého výskytu stejného stavu). Výchozí stav prvního pravidla je definován jako počáteční stav. Zkráceně zapsaný konečný automat obsahuje právě všechny stavy a symboly použité v seznamu pravidel. Všechny komentáře jsou zahazovány. Příklad:
```
start 'a' -> start, start 'b' -> stav2, stav2'a' -> stav2,#start je počáteční
stav2 'b'-> stav1 ,stav1'a'-> stav1 ,stav1 'b'->start.    #i koncový stav
```

- **MST** (0,5 bodu): Po zadání bonusového parametru `--analyze-string="retezec"` (nelze kombinovat s `-m` nebo `-f`) provede algoritmus analýzu, zda je zadaný `retezec` řetězcem jazyka přijímaného zadaným konečným automatem. Pokud ano, vypíše na výstup bez odřádkování pouze číslici `1`, jinak vypíše `0`. POZOR! Návratový kód je v případě správné funkčnosti skriptu v obou případech `0`. V případě, že `retezec` obsahuje znak, který chybí ve vstupní abecedě, dojde k chybě s kódem 1.

- **MWS** (1,5 bodu): Po zadání bonusového parametru `--wsfa` bude skript akceptovat i obyčejný deterministický konečný automat[11] (neskončí s chybou) a transformovat jej na dobře specifikovaný konečný automat (pomocí algoritmu[12] ze 4. přednášky IFJ na snímku 35/36; případný maximálně jediný neukončující stav bude mít identifikátor[13] `qFALSE`). Tento automat pak bude skript dále zpracovávat dle dalších zadaných parametrů, jakoby se jednalo o automat načtený ze vstupu.

Reference:

- A. Meduna, R. Lukáš, Z. Křivka. *Přednášky předmětu Formální jazyky a překladače (IFJ): Kapitola IV. Speciální typy konečných automatů.* FIT VUT v Brně, 2015. [cit. 2016-02-09]. Dostupné z: `https://www.fit.vutbr.cz/study/courses/IFJ/private/prednesy/Ifj04-cz.pdf`

- A. Meduna, R. Lukáš, Z. Křivka. *Přednášky předmětu Formální jazyky a překladače (IFJ): Kapitola XI. Vlastnosti regulárních jazyků.* FIT VUT v Brně, 2015. [cit. 2016-02-09]. Dostupné z: `https://www.fit.vutbr.cz/study/courses/IFJ/private/prednesy/Ifj11-cz.pdf`

[10] Tedy, výchozí stav a neprázdný vstupní symbol bez apostrofů musí být odděleny alespoň jedním bílým znakem.
[11] Předpokládejte, že vstupní konečný automat nemá stav `qFALSE`.
[12] Je-li počáteční stav neukončující, bude novým počátečním stavem stav `qFALSE`.
[13] Při kombinaci `--wsfa` s parametrem `-i` bude případně na výstupu neukončující stav `qfalse`.

3 Specifické požadavky na dokumentaci

Popište techniku ověřování lexikální a syntaktické správnosti vstupu. V případě minimalizace nepopisujte obecný algoritmus, ale specifika Vaší konkrétní implementace.

4 Poznámky k hodnocení

Výsledný automat bude nejprve porovnán na přesnou shodu pomocí nástroje `diff`. V případě neúspěchu bude s bodovou srážkou provedena normalizace jinak syntakticky správného výstupního konečného automatu a uskutečněno nové porovnání.