Dokumentace úlohy MKA: Minimalizace konečného automatu v Python 3 do IPP 2016/2017
Jméno a příjmení: Jakub Frýz
Login: xfryzj01

Úvod

Následující text popisuje proces skriptu

1 Vstupní argumenty

Vstupní argumenty se zpracovávají pomocí `argparse`. Kontroluje se:

- je-li zadán argument `--help` a nic navíc:

  ano vypíše se nápověda

  ne CHYBA

- jsou-li zadány oba argumenty `--find-non-finishing` (`-f`) a `--minimize` (`-m`) → CHYBA
- je-li zadán argument `--input`:

  ano načtou se data ze souboru

  ne čte se ze standardního vstupu `stdin`

- je-li zadán argument `--output`:

  ano data se uloží do souboru

  ne data se vytisknou na standardní výstup `stdout`

- je-li zadán argument `--case-insensitive` (`-i`) → po načtení souboru se celý soubor převede na malá písmena

2 Zpracování vstupního souboru

Vstupní soubor zpracovávám převážně za pomoci regexů. Na začátku odstraním komentáře a všechny bílé znaky, co nejsou mezi ". Následně pak pomocí funkce `glue()` převedu celý automat na jednořádkový `string` tzn. že zbylé bílé znaky byly převedeny na jejich „ne bílou formu“ (`\n`, `\t`, `\r`). Na takto vzniklý řetězec se zavolá funkce `process()`, která za pomoci tohoto regexu

```
^\(\{(.*)\},\{(.*)\},\{(.*)\},(.*),\{(.*)\}\)$
```

získá z onoho řetězce pětici, která definuje automat. Ta je následně pomocí dalších regexů rozkouskována na jednotlivé stavy, znaky, atd.. Ve výsledku funkce vrátí 4 `listy` (stavy, abeceda, pravidla, konečné stavy) a jeden `string` (počáteční stav). Těchto pět položek je po návratu z funkce uloženo do dalšího `listu` do proměnné jménem `automat`.

Teď je v proměnné `automat` uložen konečný automat, ale nejsme si jisti, zda-li je dobře specifikovaný. Na to je funkce `specify()`, která zkontroluje, zda-li automat splňuje podmínky:

- neobsahuje ε-přechody
- neobsahuje nedeterminismus
- neobsahuje nedostupné a neukončující stavy

Pokud podmínky splňuje funkce navrátí hodnotu `True`, jinak `False`. Pokud vrátí `False`, skript se ukončí s návratovou hodnotou `62`.

Tímto by mělo být potvrzeno, že zadaný automat je dobře specifikovaný.

Pokud byl zadán argument `--find-non-finishing` (`-f`) a nebo `--minimize` (`-m`) provedou se odpovídající funkce.

Následuje už jenom výpis automatu. Ten se provede buď na standardní výstup nebo do souboru, byl-li zadán argument `--output`.

3 Nalezení neukončujícího stavu

K tomuto účelu zde slouží funkce `find_non_finishing()`, která není nic jiného než upravená část funkce `specify()` s jediným rozdílem, že vrací název neukončujícího stavu.

4 Minimalizace konečného automatu

POZN: minimalizaci se mi nepodařilo do konce pořádně zprovoznit.

Minimalizace automatu má vytvořit minimální možnou verzi zadaného automatu. To provede funkce `minimize()`. Vytvoří se dvě množiny stavů (ty koncové a ty ostatní). Následně se prochází cyklem přes znaky abecedy a hledají se pravidla, která vycházejí ze stavů jedné a pak i druhé množiny samostatně. Pokud všechna nalezená pravidla pro daný znak i množinu stavů končí ve stavech, která jsou všechny součástí jedné množiny, nic se neděje, pokud ovšem se stane, že některé jsou z druhé množiny, provede se štěpení a štěpí se tak dlouho, dokud se už nedá více štěpit.

Pak dojde ke spojování stavů. Projdou se všechna pravidla znovu a pokud se najdou taková pravidla, která přes stejný znak vycházejí ze stavů z jedné množiny a končí v jiné či stejné, lze tyto pravidla nahradit spojenou verzí.

Závěr

Tenhle projekt byl pro mne mnohem těžší než předchozí. Jedním z důvodů může příliš volný styl kódu jazyka Python3. Osobně preferuji, když jsem syntaxí jazyka trochu „omezován“ jako například znaky { a } pro funkce, cykly, podmínky, atd.. Osobně mi tento styl přijde přehlednější. Ale zde už záleží na vkusu člověka, který ten jazyk používá. I tak jsem se mnohé přiučil, ať už z jazyka Python3, ale i tím, že jsem si mohl vyzkoušet algoritmizaci už známých procesů z předmětu IFJ.