Dokumentace úlohy DKA: Determinizace konečného automatu v PHP 5 do IPP 2013/2014do IPP 2016/2017

Jméno a příjmení: Václav Martinka

Login: xmarti76

1. Obecné řešení zadání

Při řešení jsem využil zejména zkušenosti z loňského předmětu IFJ, konkrétně s programováním lexikálního analyzátoru. To se projevilo na implementaci načítacích funkcí, které jsou postavené na principu konečného automatu. Co se týče přehlednosti a délky kódu, tak by jistě bylo vhodnější využít některé z pokročilejších stringových funkcí jazyka PHP 5. Na druhou stranu mnou zvolené řešení by mělo být méně náročné a více odolné vůči nevalidním vstupům.

Lexikální analýzu zde přímo doplňuje syntaktická analýza. Díky tomu, že pravidla pro vstup jsou jasně daná, mohu si dovolit načítat vstup po znacích a rovnou je kontrolovat, zda spadají do množiny znaků, kterou mohu v této části vstupu očekávat.

O správné načtení algoritmů se stará funkce `reag_args()` ze souboru `read_args.php`. Ta je postavena na principu `switch(string argv[i])`. Ukládá načtené informace do struktury `options` (viz níže) a zároveň provádí jejich validaci.

2. Struktura skriptu

Celý skript je postaven na dvou rozsáhlejších třídách, a to `options` a `finite_automaton`, umístěných v souboru `classes.php`.

- **Třída `options`**

  Tato třída slouží pro uchování hodnot načtených z příkazové řádky, jedné se vlastně jen o společný „obal“ pro jednotlivé proměnné. Obsahuje jen 4 metody, 2 pro inicializaci vstupu/výstupu, 1 pro test na `EOF` a poslední nejdůležitější `get_char()`, která načtu buď následující vstupní znak nebo až následující platný vstupní znak (ignoruje bílé znaky a koemntáře).

- **Třída `finite_automaton`**

  Jak je z názvu patrné, jedná se třídu reprezentující konečný automat. Obsahuje množinu stavů (každý stav je objektem třídy `state` a uchovává si kromě svého ID i informaci o tom, zda se jedná o koncový stav), množinu pravidel (třída `rule`), množinu vstupních symbolů a ukazatel na počáteční stav. Množiny jsou řešeny pomocí polí.

  Dále jsou zde metody `get` pro čtení množiny, `add` pro přidání objektu do množiny, `is` pro ověření existence v množině a metoda `__toString()` která má na starosti vypsaní všech hodnot konečného automatu ve správném tvaru

- **Soubor `functions.php`**

  Do tohoto souboru jsem umístil zbytek potřebných funkcí. Zejména pětici funkcí, které se starají o naplnění konečného automatu daty ze vstupu metodou zmíněnou na začátku.

  Dále jsou zde umístěny funkce pro odstranění ε-přechodů, determinizaci a analýzu stringu.

  Všechny tyto funkce jsou ve vhodném pořadí volány ze souboru `dka.php`.

3. Implementace algoritmů IFJ

- **Odstranění ε -přechodů**

Na začátku si vytvořím nový objekt konečného automatu a zkopíruji do něj některé hodnoty z původního (množinu stavů (včetně informace o koncovém stavu), vstupní abecedu a počáteční stav).

Pro každé pravidlo si vytvořím jeho ε-uzávěr a jakmile ho mám, tak rovnou vytvořím nové pravidlo (pravidla) dle algoritmu z IFJ. Tyto pravidlo zároveň ukládám do mého nově vytvořeného konečného automatu.

Jakmile projdu všechny stará pravidla a uložím všechna nová, přepíšu referenci na konečný automat a ukončím funkci.

- **Odstranění nedeterminizmu**

  Funkci je nutné volat nad konečným automatem bez ε-přechodů.

  Na začátku si opět vytvořím nový stavový automat a zkopíruji do něj abecedu a vstupní stav z původního.

  Opět dodržuji algoritmus z IFJ, tedy že vezmu nezpracovaný stav (na začátku mám jen jeden), postupně projdu vstupní abecedu a množinu pravidel a vytvářím nová pravidla (je-li to potřeba, tak taktéž i stavy). Po té, co zpracuji všechny stavy, tak opět přepíšu referenci a ukončím funkci.

4. Rozšíření

I když funkce `read_args()` akceptuje všechny možná rozšíření, tak nakonec jsem implementoval jen jedno a to STR. Nejdřív je nutné provést determinizaci. Poté lze již spustit tuto funkci.

- **SRT**

  Jedná se o jednoduchý průchod pravidly konečného automatu na základě aktuálního stavu a aktuálního symbolu na vstupu. Na začátku se nacházím v počátečním stavu a na vstupní symbol odpovídá prvnímu znaku v kontrolovaném stringu. Provedu kontrolu, zda se nachází ve vstupní abecedě, pokud ano, projdu všechna pravidla pro aktuální stav a hledám pravidlo pro aktuální symbol. Pokud se mi ho podaří najít, pokračuji na další symbol. Až takto projdu celý vstupní string, zkontroluji, že jsem v jednom z koncových stavů a pak na rozdíl od předchozích funkcí nevracím referenci na konečný automat, ale string „`1`“. V případě, že by jakákoli z podmínek nebyla splněna, vracím „`0`“.