Dokumentácia úlohy DKA: determinizácia konečného automatu v PHP5 do IPP 2015/2016
Meno a priezvisko: Adam Ormandy
Login: xorman00

Dokumentácia projektu IPP, variant DKA

Kontrola príkazovej riadky

Prvou činnosťou skriptu je načítanie a kontrola argumentov príkazovej riadky. Načítanie sa uskutočnuje pomocou funkcie `array` **`getopt`**`(string $options[,array $longopts])`. Táto funckia vracia pole obsahujúce argumenty CLI.

Po načítaní prebieha nastavenie premenných ktoré ovplyvňujú ďalšie chovanie skriptu.
Ak je zapnutý prepínač `–help`, tak sa ihneď zobrazí pomoc a skript sa ukončí. Ďalej prebieha aj kontrola prepínačov `-e` a `-d`, ktoré nemôžu byt zapnuté súčasne. Ak sú zapnuté súčasne, skript sa ukončí.

Načítanie a kontrola vstupného automatu

Druhou činnosťou je načítanie a kontrola vstupného automatu. Ak je zapnutý prepínač `--input`, tak načítanie KA prebieha zo špecifikovaného súboru. Inak načítanie prebieha z `stdin`. V prípade problému z načítaním súboru, resp. `stdin`, sa skript ukončí s návratovou hodnotou 2. Zároveň, ak je zapnutý prepínač `–case-insensitive`, všetky veľké písmená prepíšeme malými pomocou `mb_strtolower`.

Po načítaní vstupného automatu prebieha jeho kontrola a zapísanie do štruktúr, pre budúce spracovanie. Spracovanie prebieha v niekoľkých fázach.

V prvej fáze sa odstráni od-riadkovanie a komentáre. Tu sa aplikuje jedna výnimka a to v prípade znaku od-riadkovania ako symbol abecedy. Tento problém som vyriešil substitúciou `'\n'` za `'<>'`. Po odstránení od-riadkovania a komentárov nahradím `'<>'` za `'\n'`.

V druhej fáze odstránim medzery a tabulátory. Tu sa tiež aplikuje podobná výnimka ako pri `'\n'`, lebo zo zadania vyplýva, že aj tabulátor a medzera môžu byť symbolom abecedy.

Počas tretej fázy sa skontroluje výraz sa skontrolujú zátvorky na konci automatu a automat sa rozdelí na jednotlivé pod-komponenty, teda na stavy, abecedu, pravidlá, začiatočný stav a koncové stavy.

Štvrtá fáza sa skladá z kontroly a zápisu vstupnej abecedy a stavov. Kontrola stavov prebieha pomocou regexu, ktorý kontroluje, či daný zápis je platným identifikátorom jazyka C. Ďalej sa kontroluje či nie je tento nový stav duplicitný zo stavom, ktorý sme už načítali. Ak je, tak sa nezapisuje a pokračuje sa ďalej v spracovaní.

Vstupná abeceda bol mierny oriešok, a to hlavne z dôvodu výnimiek. Vstupnú abecedu najprv rozdelím na jednotlivé komponenty, teda symboly. Potom spracovávam jednotlivé symboli. Ak daný symbol už je v abecede, tak ho neukladám znova, teda sa bránim proti duplicitám. Pri spracovaní abecedy môže nastať niekoľko výnimiek, a to pri spracovaní symbolov `','''` a `''''`, ktoré riešim osobitými pravidlami, ktoré sa nachádzajú na riadkoch 240 až 255.

Piata fáza, spracovanie prechodových pravidiel, je najzaujímavejšou fázou. Tu sa taktiež pomocou regulárneho výrazu najprv rozdelí množina pravidiel na jednotlivé pravidlá. Následne sa pravidlá rozdelia na jednotlivé pod-komponenty, teda vstupný stav, symbol a výsledný stav. Kontrola má 2 stupne, najprv sa skontroluje formálny zápis pravidla, ak je ten vyhovujúci, kontrolujeme jednotlivé pod-komponenty. Pri stavoch kontrolujeme či je daný stav z množiny stavov a pri symbole, či je daný symbol v množine abeceda. Ak jeden z nich nie je v danej množine, ukonči sa skript z chybovým kódom 41.

Ako posledné sú fázy kontroly a spracovania vstupného stavu a množiny koncových stavov. Tu sa nekontroluje či je daný stav platným identifikátorom jazyka C, lebo takáto kontrola by bola zbytočná, ale len to, či je daný stav podmnožinu stavov konečného automatu.

Odstránenie epsilon prechodov

Na odstránenie epsilon prechodov používam techniku epsilon uzáveru, ktorá bola popísaná v na predmete IFJ.

Najprv si vytvorím pole `$new_rules`, do ktorého budem ukladať nové pravidlá, bez epsilon prechodov.

Následne začína cyklus prechádzania jednotlivých stavov. V každom cykle sa robí epsilon uzáver jedného stavu. Tu používam dve polia `$processed_states` a `$states_to_go`. `$processed_states` obsahuje už spracované stavy pri danom epsilon uzávere, a zabraňuje zbytočnemu analyzovaniu stavov alebo zacykleniu.

Dokumentácia úlohy DKA: determinizácia konečného automatu v PHP5 do IPP 2015/2016
Meno a priezvisko: Adam Ormandy
Login: xorman00

`$states_to_go` obsahuje stavy ktoré ešte len v rámci vytvárania epsilon uzáveru musíme prejsť. Implicitne je prvý člen tohto poľa stav, ktorého epsilon uzáver práve robíme. V poli `$states_to_go` sa orientujeme vďaka indexu `$index_states_to_go.`

Uzáver robím postupným prechádzaním pravidiel. Ak má pravidlo ako vstupný stav, stav na ktorý nám ukazuje `$index_states_to_go` tak ho bližsie spracujeme. Ak to je pravidlo bez epsilon prechodu, tak toto pravidlo dáme do poľa `$new_rules,` kde ako vstupný stav dáme stav, ktorého epsilon uzáver spracovávame. Ak je to pravidlo z epsilon prechodom, tak konečný stav tohto pravidla dáme do pola `$states_to_go`. Keď prejdeme všetky pravidlá, zväčšíme `$index_states_to_go` o 1.

Epsilon uzáver daného stavu je hotový, ak `$index_states_to_go` je rovný počtu prvkov poľa `$states_to_go.`

Po vytvorení epsilon uzáveru pre všetky stavy, nahradíme staré pravidlá novými z poľa `$new_rules` a z abecedy vylúčime epsilon prechod.

Determinizácia

Determinizácia začína vytvorením polí `$new_states, $new_rules, $new_end_states` a `$heap_states`. Prvé tri nám simulujú nový deterministický automat ktorý vytvárame. `$heap_states` sa správa ako kopa, do ktorej ukladáme všetky novo nájdené, nepreskúmané stavy. Implicitne je na začiatku determinizácie v poli `$heap_states` len počiatočný stav.

Proces determinizácie prebieha takto, najprv vyberieme z kopy `$heap_states` jeden prvok, ktorý dáme do `$processed_state`. Skontrolujeme či už sme daný prvok nepreskúmavali, ak nie, pokračujeme ďalej.

Začneme cyklus v ktorom prechádzame jednotlivé symboly abecedy. V tomto cykle začneme pod-cyklus a vytvoríme pole `$out_state,` do ktorého budeme ukladať združený stav, ktorý vzniká pri spájaní stavov.
Pod-cyklus prechádza jednotlivé pravidlá, ak má dané pravidlo počiatočný stav rovný `$processed_state` a symbol je zhodný zo symbolom ktorý skúmame, tak koncový stav daného pravidla pridáme do poľa `$out_state.`

Po prejdený všetkých pravidiel overíme, či je aspoň jeden stav v `$out_state`. Ak je tam jeden alebo viac, tak tieto stavy združíme pod novým identifikátorom stavu. Ak nebol stav z takýmto identifikátorom už preskúmaný, tak ho uložíme do poľa `$new_states a $heap_states.` Zároveň zapíšeme novo vzniknuté pravidlo v tvare `$processed_state 'symbol' -> $out_state` do poľa `$new_rules.`

Po skončení determinizacie pre daný `$processed_state` ešte skontrolujeme, či nebol nájdený nový ukončujúci stav. Ak sme našli nový ukončujúci stav, tak ho zapíšeme do `$new_end_states.`

Takto prehľadávame všetky stavy v `$heap_states`, pokým už tam nebudú žiadne stavy k prehľadaniu. Ak nastane koniec determinizácie, tak sa starý automat prepíše novým, a tento nový ide na výpis.