Generátor umožňující rychlé prototypování překladačů SLR(1) jazyků

Autorizovaný software

Jazyk sémantických podprogramů

Ing. Jiří Zuzaňák

10. prosince 2009

1 Úvod

Autorizovaný software reprezentovaný generátorem překladačů využívá při testování navrhovaného překladače, pro popis sémantických operací jednoduchý skriptovací jazyk. Tento jazyk je jen několik úrovní nad jazykem symbolických adres a umožňuje návrháři překladače přístup k datům ze zdrojového řetězce (řetězec nad kterým se navrhovaný překladač testuje).

Popisovaný jazyk byl navržen a implementován pomocí zmiňovaného generátoru překladačů. Popis jeho struktury je součástí generátoru a na základě direktiv překladače v kódu generátoru je možné nastavit zda se překladač sémantických programů generuje znovu, nebo zda se načítá z uloženého zdroje (binární reprezentace překladače).

Jazyk pracuje se základními datovými typy, umožňuje obvyklé řízení programu, definuje základní dynamické struktury (pole, zásobník) a umožňuje zápis na standardní výstup.

2 Popis jazyka a úvod do základní syntaxe

V této části textu bude popsána základní syntaxe jazyka pro popis sémantických podprogramů navrhovaného překladače.

2.1 Komentáře v jazyce

Komentáře se v jazyce zapisují pomocí symbolů `#` a `$`, které deklarují že vstup do konce řádku je interpretován jako komentář. Komentáře nejsou v podstatě součástí překladače jazyka sémantických programů, ale odstraňují přímo při rozkladu řetězce popisujícího strukturu překladače.

2.2 Základní výrazy

Podobně jako v jiných jazycích se výrazy zapisují v infixové notaci, tj. operátory se umísťují mezi operandy a interpretují se s ohledem na jejich vestavěnou prior-

itu. Pořadí vyhodnocování těchto operátorů je možné ovlivnit jejich uzavřením do závorek.

2.3 Řízení toku programu

Základní větvení programu se provádí pomocí testováni výsledku výrazu standardním příkazem `if`. Příklad jednoduché podmínky je zobrazen v následujícím bloku kódu.

```
if value == 10
   out("value is equal to ten\n");
else
   out("value is not equal to ten\n");
fi
```

Základní cyklus je možné nadefinovat pomocí klíčových slov `do` a `while`. Následuje příklad jednoduché smyčky.

```
int value = 0;
do
   out("value: ",value,"\n");
while ++value < 100;
```

V rámci zmíněných příkazů pro řízení toku programu je možné v těle podmínek a cyklů použít posloupnost příkazů (tj. shlukovat tyto příkazy do bloků). Tento postup při vytváření bloků příkazů je jediným možným způsobem jak tyto bloky v popisovaném jazyce vytvořit.

2.4 Deklarace a definice proměnných

Každou proměnnou je nutné před prvním použitím deklarovat a případně inicializovat její hodnotu. Deklarace proměnné se provádí podobně jako v jazyce `C/C++`. První je uveden typ proměnné, za kterým nasleduje její jméno. Proměnnou je možné inicializovat hodnotou přímo v její deklaraci. Deklarace proměnných reprezentujících dynamické pole se provádí stejným způsobem jako deklarace jiných datových typů (viz následující příklad).

```
int          var_i = 1;
string       var_s = "One";
int[]        var_ia = array(1,2,3),
             var_ia = empty();
string[]     var_sa = array("One","Two","Three");
int[][]      var_iaa = array(
                array(1,2,3),
                array(4,5,6)
             );
string[][]   var_saa = array(
                array("One","Two","Three"),
                array("Four","Five","Six")
             );
```

Jazyk pracuje se dvěma základními datovými typy, kterými jsou celé číslo `int` a řetězec `string`. Nad oběma těmito datovými typy jsou definovány rozšířené datové typy - dynamické pole a dynamické pole jehož prvky tvoří zmíněná (jednoprostorová) dynamická pole.

Datový typ `int`

Implementace datového typu `int` je velice podobná implementaci v jazyce `C/C++`. Celé čísla jsou používány jako náhrada booleovského typů, ve významu var $\neq 0 \Rightarrow$ true. Výsledkem testu pravdivosti výrazu je tedy prvek z množiny $\{0, 1\}$. Datový typ `int` je možné v rámci kódu sémantického programu zapsat v osmičkové, decimální a hexadecimální soustavě, nebo také jako znak uzavřený v uvozovkách (hodnota celočíselné proměnné je poté rozhodnuta `ASCII` hodnotou předaného znaku).

Datový typ `string`

Slouží pro práci s řetězci znaků. Definuje základní operace jako jsou konkatenace řetězců, nalezení podřetězce a další. Slouží primárně pro zpracování jmen terminálních symbolů (jména proměnných, funkcí, ...) navrhovaného překladače. K tomuto účelu slouží příkaz `get_rule_body` umožňující získat řetězec odpovídající zvolenému terminálnímu symbolu.

Datový typ `array`

Popis datového typu `array` v sobě zahrnuje všechny čtyři datové typy pole, které je možné v jazyce použít (`int[]`, `string[]`,`int[][]`, `string[][]`). Konstantní pole se definuje pomocí příkazu `array` a `empty`, kde první varianta vytvoří pole na základě předaných parametrů zatímco druhá vytvoří prázdné pole.

2.5 Další funkce a příkazy

Následuje krátký popis některých funkcí vestavěných do jazyka sémantických podprogramů za účelem snadnějšího testování navrhovaného překladače.

- `out(<any>,...)` - vytiskne hodnotu všech předaných argumentů v zadaném pořadí na standardní výstup
- `assert(int)` - otestuje logickou hodnotu předaného parametru a pokud je nepravdivá vytiskne na standardní výstup chybovou zprávu obsahující řádek na kterém se příkaz vyskytuje, následovaný řetězce reprezentujícím testovanou podmínku
- `substr(string,int,int)` - vytvoří podřetězec na základě obsahu řetězce reprezentovaného prvním parametrem. Výsledný řetězec se v původním řetězci nachází na pozici určené druhým parametrem a má délku definovanou třetím parametrem
- `format(string,int[])` - formátuje tzv. escape sekvence v řetězci popsaném prvním argumentem.
- `rule_body(int)` - získá řetězec popisující terminální symbol v pravidle u kterého se tento sémantický kód nachází na pozici určené parametrem příkazu.
- `line_cnt()` - získá číslo řádku rozkládaného kódu na kterém se provádí redukce pravidla, k němuž je přiřazen sémantický kód obsahující tento příkaz.

- `to_int(string)` - zkonvertuje řetězec na celé číslo.
- `to_string(int)` - zkonvertuje celé číslo na řetězec.
- `empty()` - vytvoří prázdné pole, přiřaditelné do pole libovolného typu.
- `array(<array_type>, ... )` - vytvoří konstantní pole, jehož typ se rozhodne podle předaných argumentů. Argumentem nesmí být vícerozměrné pole.
- `size(string|<array>)` - získá velikost řetězce, nebo aktuální počet prvků v poli.
- `push(<array>,<value>)` - vloží na konec pole popsaného prvním argumentem, hodnotu danou druhým argumentem.
- `pop(<array>)` - Vyzvedne z pole prvek nacházející se na jeho konci. Pole musí obsahovat nejméně jeden prvek.
- `remove(<array>,int)` - odstraní z konce pole popsaného prvním argumentem počet prvků určený druhým argumentem. Podobně jako u příkazu `pop` je nutné aby pole obsahovalo minimálně takový počet prvků, který se požaduje k odstranění.
- `get_idx(<array>,<array_type>)` - nalezne v poli obsaženém v prvním argumentu pozici prvního prvku, který má stejnou hodnotu jako druhý argument příkazu, pokud tento prvek v poli není obsažen vrátí příkaz hodnotu -1.
- `get_next_idx(<array>,<array_type>,int)` - plní stejnou funkci jako příkaz `get_idx` s tím rozdílem že vyhledává prvek od pozice určené třetím argumentem.

3 Závěr

Jazyk pro popis sémantických podprogramů je navržen za účelem jednoduchého testování správnosti rozkladu vstupního řetězce navrženou gramatikou a zpracování řetězců popisujících terminální symboly tohoto překladače. Jazyk je jednoduchý a neumožňuje složitější konstrukce, jakými jsou například definice funkcí, tříd a jejich metod a podobné.

Jazyk je doporučeno používat především jen jako ladící nástroj při návrhu gramatiky a regulárních výrazů konstruovaného překladače.