Generátor umožňující rychlé prototypování překladačů SLR(1) jazyků

Autorizovaný software

Dokumentace

Ing. Jiří Zuzaňák

10. prosince 2009

Abstrakt

V tomto dokumentu bude popsán program a jeho zdrojový kód, vykazovaný v roce 2009 jako autorizovaný software. Popisovaný software je navržen a implementován za účelem rychlého návrhu a prototypování překladačů zdrojových kódů zapsaných v jazycích spadajících do třídy jazyků zpracovatelných `SLR(1)` překladači. Překladač je generován na základě textového popisu, který definuje tvary terminálních symbolů jazyka a jeho základní gramatiku. Generátor umožňuje okamžité testování překladače definovaného výše popsaným způsobem na zadaném vstupním řetězci, včetně aplikace jednoduchých sémantických programů přiřazených k pravidlům gramatiky.

Autorizovaný software je určen ke stažení bez jakýchkoli omezení (viz. licence), s tím že při použití zdrojových kódů je nutné zachovat jméno autora a instituce.

1 Úvod

V dnešní době se již překladače a na nich založené kompilátory nesestavují ručně. Při jejich tvorbě je používáno poznatků z teorie formálních jazyků a překladačů umožňující, vysokou úroveň automatizace návrhu překladače na základě jeho formálního popisu. Zmiňovaný formální popis jazyka se skládá ze dvou základních částí, kterými jsou: popis nedělitelných stavebních prvků jazyka - terminální symboly a popis syntaxe jazyka - gramatika.

Terminální symboly bývají ve většině podobných nástrojů (generátorech překladačů) často popisovány regulárními výrazy přímo těžícími z teorie formálních jazyků - konečné automaty, jejich paralelizace a generování ekvivalentních regulárních výrazů. Výsledkem generování automatu rozpoznávajícího terminální symboly na základě jejich popisu je lexikální analyzátor. Mezi známější nástroje pro automatické generování lexikálního analyzátoru patří například `lex` a `flex`.

Druhou součástí popisu jazyka je popis jeho syntaxe pomocí gramatiky, zapsané v některé z normálních forem. Často se volí `BNF` (Backus-Naur) normální forma. Stejně jako v předchozím případě (terminálních symbolů) je generování

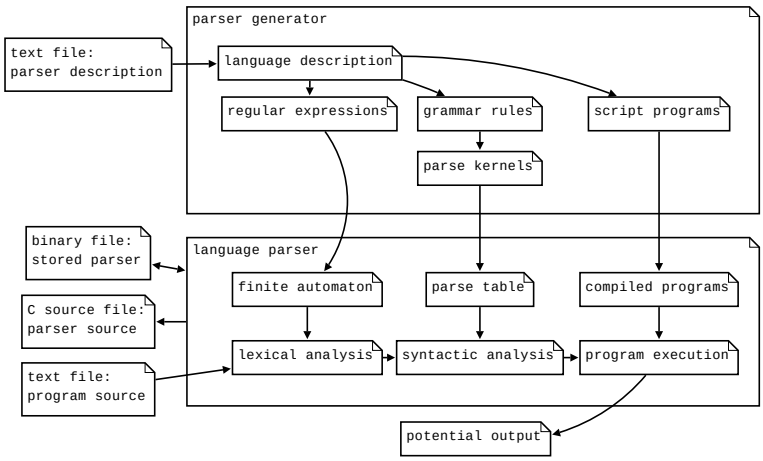
syntaktického analyzátoru založeno na rozsáhlé teorii formálních jazyků a jí popsaných algoritmech automatizujících tvorbu rozkladových tabulek z gramatiky jazyka. Mezí známější nástroje generující syntaktický analyzátor jazyka patří například `yacc` a `bison`.

Překladač vygenerovaný programem (reprezentující autorizovaný software) na základě zmiňovaného popisu je reprezentován jako struktura v paměti programu, kterou je možné otestovat na předaném řetězci jazyka, nebo ji exportovat ve formě zdrojového kódu jazyka `C/C++`.

Dokument je dělen do následujících sekcí. V první sekci se nachází popis autorizovaného softwaru a samotného generátoru překladačů. V následující, v pořadí druhé sekci je popsána struktura a obsah vstupního souboru přijímaného generátorem. V třetí sekci jsou detailněji popsány sémantické podprogramy, tvořící součást popisu generovaného překladače. V následující sekci je popsána implementace generátoru a postup, kterým je možné program zkompilovat. Předposlední sekce obsahuje příklady aplikací a nástrojů, při jejichž návrhu a implementaci byl popisovaný software (generátor překladačů) použit. Dokument je ukončen závěrem ve kterém je zhodnocen přínos autorizovaného softwaru a doporučeny způsoby jeho využití.

2 Základní popis softwaru

Hlavní součástí autorizovaného softwaru je program, který na základě zadaných argumentů vygeneruje, nebo načte již existující překladač. Takto vytvořený překladač je poté možné uložit v binární formě do souboru, vygenerovat jeho zdrojový kód v jazyce `C/C++`, nebo jej použít k rozkladu zadaného vstupního řetězce. Základní struktura generátoru je zobrazena na obrázku 1.



Obrázek 1: Struktura generátoru překladačů

Program na vstupu akceptuje následující argumenty (za každým z těchto argumentů by měl následovat název souboru).

- `--parser_descr <file>` - odkaz na soubor obsahující popis překladače. Tento soubor by měl obsahovat regulární výrazy popisující terminální

symboly a samotnou gramatiku jazyka. Struktura tohoto souboru bude popsána dále v textu.

- `--parser_load <file>` - odkaz na soubor obsahující překladač uložený v binární formě. Tento je programem načten do jeho vnitřní reprezentace.
- `--parser_save <file>` - uloží aktuálně načtený, nebo vygenerovaný překladač do odkazovaného souboru. Takto uloženou reprezentaci překladače je poté možné načíst pomocí předcházejícího argumentu programu.
- `--parser_save_cc <file>` - vygeneruje `C/C++` zdrojový kód, který po přeložení vykonává rozklad překladačem popsaného jazyka. Výsledek (vygenerovaný kód) je možné upravit několika makry nastavenými při překladu generátoru (např.: způsob reprezentace konečného automatu).
- `--source <file>` - odkaz na zdrojový soubor jazyka, který je použit k otestování aktuálně načteného, nebo vygenerovaného překladače.

Po vygenerování `C/C++` souboru pomocí argumentu `--parser_save_cc` a jeho následném přeložení vznikne binární soubor, který na předaném textovém vstupu spustí rozklad podle popsané gramatiky a v jeho průběhu vypisuje na standardní výstup indexy pravidel podle kterých provádí jednotlivé redukce. Vygenerovaný zdrojový soubor obsahuje `LALR` rozkladovou tabulku a dvě základní funkce realizující rozklad zadaného řetězce. První z těchto funkcí nazvaná `final_automata_recognize` slouží k nalezení jednotlivých terminálních symbolů jazyka na základě vygenerovaného konečného automatu. Druhá základní funkce `parser_parse_source_string` řídí samotný rozklad programu na základě výše zmiňované rozkladové tabulky.

3 Popis jazyka přijímaného překladačem

Popis jazyka je rozdělen do čtyř základních sekcí, logicky jej rozdělujících podle významu. Příklad vstupu generátoru je zobrazen v Tabulce 1.

- `init_code` - sémantický kód, který se spouští před začátkem testování překladače na vstupním řetězci. Zde by měly být inicializovány všechny globální proměnné programu. Tyto je možné inicializovat i v jednotlivých sémantických podprogramech pravidel, ale to přispívá k nepřehlednosti.
- `terminals` - popisuje množinu terminálních symbolů, a přiřazuje každému z nich regulární výraz definující jeho tvar. Regulární výrazy není možné kombinovat z předešlých výrazů, tyto se musí skládat výhradně z předdefinovaných symbolů. Terminálním symbolům jsou přiřazeny speciální vlastnosti na základě tvaru jejich identifikátoru.
  - `SKIP` - terminální symbol, jehož identifikátor obsahuje podřetězec `SKIP` je přeskočen a tudíž není umístěn na zásobník v rámci syntaktické analýzy.
  - `END` - identifikátor obsahující podřetězec `END` může být použit jen jednou. Definuje terminální symbol ukončující rozklad vstupního řetězce.

```
init_code:
$ - initial code -
{
   ...
}

terminals:
   $ - set of terminal symbols, examples follows -

   single_char_const {'\''.!'\0'.'\''}
   octal_char_const {'\''.'\\'.<07>.(<07>+e).(<07>+e).'\''}
   backslash_char_const {'\''.'\\'.[abfnrtv\\?\'"].'\''}

   oct_int_const {'0'.<07>*}
   dec_int_const {<19>.d*}
   hex_int_const {'0'.[xX].(<09>+<af>+<AF>).(<09>+<af>+<AF>)*}

   id {('_'+l).('_'+l+d)*}

   class {"class"}
   extends {"extends"}

   ...

   _SKIP_ {w.w*}
   _SKIP__ {'#'.!'\n'*.'\n'}
   _SKIP___ {"//".!'\n'*.'\n'}
   _SKIP____ {"/*".(!'*'+('*'.!'/'))*."*/"}
   _END_ {'\0'}

nonterminals:
   $ - set of nonterminal symbols, examples follows -

   <start>
   <end_check>
   <program>

   ...

rules:
   $ - language grammar rules, examples follows -

   <start> -> <end_check> ->> {null}

   <end_check> -> <program> _END_ ->> {null}
   <end_check> -> _END_ ->> {null}

   <program> -> <main_class_def> ->>
   $ - rule semantic code -
   {
      ...
   }

   <def_modifier> -> public ->>
   $ - rule semantic code -
   {
      ...
   }
```

Tabulka 1: Ukázka vstupu generátoru

```
terminals:
   single_char_const {'\''.!'\0'.'\''}
   octal_char_const {'\''.'\\'.<07>.(<07>+e).(<07>+e).'\''}
   backslash_char_const {'\''.'\\'.[abfnrtv\\?\'"].'\''}
   ...
```

- `nonterminals` - množina neterminálních symbolů uzavřených v úhlových závorkách. Pokud není neterminální symbol nadefinován v této sekci a je použit při popisu pravidel, generátor reaguje chybovým hlášením.

```
nonterminals:
   <command_list>
   <command>

   <command_block>
   <command_block_begin>

   ...
```

- `rules` - sekce obsahující jádro gramatiky navrhovaného jazyka. Gramatika je popsána pravidly vyjádřenými v normální formě odvozené z `BNF`. Jednotlivé pravidla se skládají z jejich levé strany, oddělené od těla pravidla symboly `->`. Tělo pravidla se skládá z posloupnosti terminálních a neterminálních symbolů gramatiky a za ním se nachází oddělený symboly `->>` sémantický podprogram pravidla uzavřený ve složených závorkách.

```
rules:
   <rule_head> -> <list of terminals and nonterminals> ->>
   {
      ...
   }

   ...
```

Jednotlivé sémantické kódy se spouštějí pouze při testovaní navrženého překladače přímo na předaném vstupním řetězci. Při generování zdrojového `C/C++` kódu překladače nemají sémantické podprogramy na výsledek žádný vliv. Sémantiku překladače reprezentovaného zdrojovými soubory si musí návrhář překladače na-implementovat v jazyce `C/C++`.

4 Sémantické podprogramy

K jednotlivým pravidlům gramatiky je možné přiřadit tzv. sémantický kód, jehož binární reprezentace (bytový kód) se spouští při redukci podle pravidla.

Prvním spuštěným sémantickým kódem v rámci testování navrhovaného překladače na vstupním řetězci je tzv. inicializační kód. Tento kód by měl sloužit k základnímu nastavení proměnných sémantických programů. Jednotlivé sémantické podprogramy sdílejí jmenný prostor v rámci celého překladače.

Popisované sémantické podprogramy je možné využít k ladění navrhované gramatiky a regulárních výrazů, bez potřeby generování nového překladače ze zdrojového kódu v cílovém programovacím jazyce. Tato vlastnost umožňuje využít generátor k rychlému návrhu překladače zvoleného jazyka.

Příklad 1. *Jako příklad využití sémantického jazyka může sloužit vstupní soubor* `examples/script_langauge.rules`*, jehož kód generuje na základě vstupního řetězce (např.:* `examples/script_language.src`*) zdrojový kód v jazyce* `dot`*, popisující rozkladový strom předaného kódu.*

4.1 Syntaxe jazyka sémantických podprogramů

Sémantické podprogramy jsou zapsány v jednoduchém interpretovaném jazyce, který je jen o několik úrovní výše než jazyk symbolických adres. Popisovaný jazyk umožňuje návrháři překladače přistupovat k datům z rozkládaného řetězce a pracovat s základními datovými typy.

Jazyk pro popis sémantických podprogramů obsahuje konečnou množinu klíčových slov, která je zobrazena v tabulce 2. Mezi tyto slova patří i "funkce" pro práci s datovými typy (například řetězci), to proto že jsou tyto funkce přímo vestavěny do interpretu jazyka.

Zajímavostí je že tento jazyk (překladač jazyka) byl v původní verzi programu vytvářen částečně ručně (rozkladové tabulky, a generování lexikálních konečných automatů pomocí externích nástrojů), zatímco v aktuální verzi je k tvorbě již využíván popisovaný generátor překladačů. Popis jazyka sémantických podprogramů je součástí zdrojových kódů generátoru. Této metodě (sebe konstrukčnímu procesu) se říká bootstraping.

`empty`	`array`	`size`	`push`	`pop`	`remove`
`last`	`tail`	`get_idx`	`get_next_idx`	`null`	`if`
`fi`	`else`	`do`	`while`	`assert`	`substr`
`format`	`to_int`	`to_string`	`rule_body`	`line_cnt`	

Tabulka 2: Klíčové slova jazyka sémantických podprogramů

Z tabulky 3. vyplývá že jazyk sémantických podprogramů podporuje jen omezené množství datových typů. Mezi ty základní patří celé čísla a řetězec. Nad těmito základními datovými typy je možné definovat jedno-rozměrné, nebo dvou-rozměrné dynamické pole. Jmenný prostor je definován pouze jeden a k definovaným proměnným se přistupuje ze všech sémantických podprogramů stejně.

`int`	`string`
`int[]`	`string[]`
`int[][]`	`string[][]`

Tabulka 3: Základní datové typy jazyka sémantických podprogramů

Jazyk pro popis sémantických podprogramů překladače definuje několik základních funkcí pro práci s daty získanými ze zdrojového řetězce.

- `rule_body(idx)` - Vrátí řetězec rozpoznaný jako terminální symbol. Požadovaný terminál je označen indexem prvku v těle pravidla. Pokud se na zvolené pozici nachází neterminální symbol není získaný řetězec jednoznačně určitelný (bude obsahovat jeden z terminálních symbolů pravidla jehož levou stranu tvoří vybraný neterminál).
- `line_cnt()` - Vrátí pozici (řádek) na které se překladač aktuálně nachází ve zdrojovém řetězci.

Příklad 2. *Příklad jednoduchého sémantického programu, generujícího část souboru popisujícího rozkladový strom vstupního řetězce (viz. Příklad 1.).*

```
$ -- if, if-else statement --
<command> -> if <condition> <if_else> ->>
{
   if gen_parse_tree
      this_idx = node_idx++;
      out("    node_",this_idx," [label = \"if <condition> <if_else>\"]\n");
      out("    node_",this_idx," -> node_",pop(node_stack),"\n");
      out("    node_",this_idx," -> node_",pop(node_stack),"\n");
      push(node_stack,this_idx);
   else
      null
   fi
}
```

5 Implementace a překlad programu

Program je implementován v jazyce `C/C++` s použitím standardních `C/C++` knihoven operačního systému. Z výše uvedeného vyplývá že program (jeho zdrojový kód) je přenositelný na všechny platformy podporující překladač jazyka `C/C++`.

Generátor je přeložitelný pomocí `Makefile` souboru nacházejícím se u zdrojových kódů, zadáním příkazu `make`. K úspěšnému překladu je potřeba aby v systému byl nainstalován interpret jazyka `Perl`, ve kterém jsou napsány dva skripty generující šablony použité ve zdrojových kódech generátoru. Překlad probíhá ve třech krocích. V prvním kroku se spojí zdrojové soubory do jednoho kompletního zdroje, ve kterém se vygenerují části kódu na základě v něm obsažených šablon. Výsledný soubor se poté přeloží překladačem jazyka `C++`.

6 Příklady aplikací

V této sekci budou popsány příklady překladačů, k jejichž implementaci byl použit popisovaný generátor. Zdrojové soubory (popisy překladačů, a příklady vstupních řetězců) zmiňovaných aplikací se nacházejí v adresáři `examples`.

6.1 Skriptovací jazyk pro zpracování obrazu

Generátor překladače byl použit při návrhu a implementaci skriptovacího jazyka určeného pro zpracování obrazu. Syntaxe skriptovacího jazyka je podobná syntaxi jazyka `Java` s tím že některé jeho prvky jsou převzaty z jazyka `C/C++`. Jazyk je určen pro rychlé prototypování algoritmů pro zpracování obrazu, skládajících se ze základních operací, které tvoří vestavěné funkce (např.: Konvoluce, Detekce hran, Medián, Detekce `LBP` vzorů, Morfologické operace, apod.).

Soubor popisující překladač	-	`examples/script_langauge.rules`
Příklad vstupního řetězce	-	`examples/script_langauge.src`

6.2 Generátor šablon (`C++ STL` alternativa)

Generátor byl použit při tvorbě aplikace generující `C/C++` kód z šablon zapsaných ve zdrojovém kódu. Umožňující tak generické programování podobné jako nabízí `STL` knihovna jazyka `C/C++`. Generátor byl použit pro rozklad a interpretaci šablon reprezentovaných podřetězci zdrojového kódu.

Soubor popisující překladač	-	`examples/mt_automaton.rules`
Příklad vstupního řetězce	-	`examples/mt_automaton.src`

6.3 Překladač jazyka `dot` - načítání grafů

Jazyk `dot` slouží k popisu obecných grafů v rámci prostředí `graphviz` sloužícího k vizualizaci (nebo výpočtu rozvržení) grafů. Generátor byl použit při návrhu aplikace umožňující načítání takto popsaných grafů za účelem následného zpracování grafovými gramatikami.

Soubor popisující překladač	-	`examples/dot_parser.rules`
Příklad vstupního řetězce	-	`examples/dot_parser.src`

6.4 Překladač jazyka pro popis grafové gramatiky

Stejně jako v předcházejícím případě byl generátor použit k načítání textového řetězce popisujícího grafovou gramatiku, složenou z grafových přepisovacích pravidel, jejich vztahů a návazností.

Soubor popisující překladač	-	`examples/gg_rule_loader.rules`
Příklad vstupního řetězce	-	`examples/gg_rule_loader.src`

6.5 Organizátor

Posledním příkladem aplikace generátoru překladačů k vytvoření překladače jednoduchého jazyka je kód rozkládající vstupní soubor aplikace generující organizační struktury a tabulky vhodné pro plánování jednoduchých činností. Rozkládaný vstupní soubor se skládá z jednoduchých zanořovaných bloků a jejich parametrů.

Soubor popisující překladač	-	`examples/organizer.rules`
Příklad vstupního řetězce	-	`examples/organizer.src`

7 Závěr

I přes to že výše popsaný generátor překladačů je omezen na generování překladačů typu `SLR(1)`, je díky svým vlastnostem umožňujícím rychlé testování navrhovaného jazyka vhodný pro návrh jednodušších jazyků. Použití generátoru při návrhu a implementaci skriptovacího jazyka, jehož struktura je podobná (až na několik vyjímek téměř stejná) jako struktura jazyka `Java` naznačuje že množina jazyků popsatelná generovnatelnými překladači není tak moc omezená jak by se mohlo na první pohled zdát.