

TDT4205 Problem Set 2

Spring 2016

Answers are to be submitted by *It's Learning*, by Feb. 15th, 20:00.

1 Top-down parsing

1.1 LL(1) form

Rewrite the following grammar into LL(1) form, by left factoring it and eliminating left recursion.

$$S \rightarrow sCT|sCTwB$$
$$C \rightarrow c$$
$$T \rightarrow t|\epsilon$$
$$B \rightarrow Ba|a$$

1.2 Parsing table

Tabulate the FIRST and FOLLOW sets of the nonterminals in the resulting grammar, and construct the predictive parsing table.

2 VSL specification

The directory in the code archive ps2 skeleton.tgz begins a compiler for a slightly modified 64-bit version of VSL (“Very Simple Language”), defined by Bennett (*Introduction to Compiling Techniques*, McGraw-Hill, 1990).

Its lexical structure is defined as follows:

- *Whitespace* consists of the characters `'\t'`, `'\n'`, `'\r'`, `'\v'` and `' '`. It is ignored after lexical analysis.
- *Comments* begin with the sequence `'//'`, and last until the next `'\n'` character. They are ignored after lexical analysis.
- *Reserved words* are FUNC, BEGIN, END, RETURN, PRINT, CONTINUE, IF, THEN, ELSE, WHILE, DO, and VAR.
- *Operators* are assignment (`:=`), the basic arithmetic operators `'+'`, `'-'`, `'*'`, `'/'`, and relational operators `'='`, `'<'`, `'>'`.
- *Numbers* are sequences of one or more decimal digits (`'0'` through `'9'`).
- *Strings* are sequences of arbitrary characters other than `'\n'`, enclosed in double quote characters `""`.
- *Identifiers* are sequences of at least one letter followed by an arbitrary sequence of letters and digits. Letters are the upper- and lower-case English alphabet (`'A'` through `'Z'` and `'a'` through `'z'`), as well as underscore (`'_'`). Digits are the decimal digits, as above.

The syntactic structure is given in the context-free grammar on the last page of this document.

Building the program supplied in the archive ps2_skeleton.tgz combines the contents of the `src/` subdirectory into a binary `src/vslc` which reads standard input, and produces a parse tree.

The structure in the `vslc` directory will be similar throughout subsequent problem sets, as the compiler takes shape. See the slide set from the PS2 recitation for an explanation of its construction, and notes on writing Lex/Yacc specifications.

2.1 Scanner

Complete the Lex scanner specification in `src/scanner.l`, so that it properly tokenizes VSL programs.

2.2 Tree construction

A `node_t` structure is defined in `include/ir.h`. Complete the auxiliary functions `node_init`, and `node_finalize` so that they can initialize/free `node_t`-sized memory areas passed to them by their first argument. The function `destroy_subtree` should recursively remove the subtree below a given node, while `node_finalize` should only remove the memory associated with a single node.

2.3 Parser

Complete the Yacc parser specification to include the VSL grammar, with semantic actions to construct the program's parse tree using the functions implemented above. The top-level production should assign the root node to the globally accessible `node_t` pointer `'root'` (declared in `src/vslc.c`).

$program \rightarrow global_list$
 $global_list \rightarrow global \mid global_list \ global$
 $global \rightarrow function \mid declaration$
 $statement_list \rightarrow statement \mid statement_list \ statement$
 $print_list \rightarrow print_item \mid print_list \ , \ print_item$
 $expression_list \rightarrow expression \mid expression_list \ , \ expression$
 $variable_list \rightarrow identifier \mid variable_list \ , \ identifier$
 $argument_list \rightarrow expression_list \mid \epsilon$
 $parameter_list \rightarrow variable_list \mid \epsilon$
 $declaration_list \rightarrow declaration \mid declaration_list \ declaration$
 $function \rightarrow FUNC \ identifier \ (\ parameter_list \) \ statement$
 $statement \rightarrow assignment_statement \mid return_statement$
 $statement \rightarrow print_statement \mid if_statement$
 $statement \rightarrow while_statement \mid null_statement \mid block$
 $block \rightarrow BEGIN \ declaration_list \ statement_list \ END$
 $block \rightarrow BEGIN \ statement_list \ END$
 $assignment_statement \rightarrow identifier \ : \ = \ expression$
 $return_statement \rightarrow RETURN \ expression$
 $print_statement \rightarrow PRINT \ print_list$
 $null_statement \rightarrow CONTINUE$
 $if_statement \rightarrow IF \ relation \ THEN \ statement$
 $if_statement \rightarrow IF \ relation \ THEN \ statement \ ELSE \ statement$
 $while_statement \rightarrow WHILE \ relation \ DO \ statement$
 $relation \rightarrow expression \ = \ expression$
 $relation \rightarrow expression \ < \ expression$
 $relation \rightarrow expression \ > \ expression$
 $expression \rightarrow expression \ + \ expression$
 $expression \rightarrow expression \ - \ expression$
 $expression \rightarrow expression \ * \ expression$
 $expression \rightarrow expression \ / \ expression$
 $expression \rightarrow \ - \ expression$
 $expression \rightarrow (\ expression \)$
 $expression \rightarrow number \mid identifier \mid identifier \ (\ argument_list \)$
 $declaration \rightarrow VAR \ variable_list$
 $print_item \rightarrow expression \mid string$
 $identifier \rightarrow IDENTIFIER$
 $number \rightarrow NUMBER$
 $string \rightarrow STRING$