Project 2: Tiger Language LL(1) Parser

Dong Son Trinh
Maxim Mints

High-Level Parser Architecture:

The parser makes use of the scanner created for Project 1. That scanner, capable of producing lists of words for parsers requiring length-k lookahead (architectural decision made for potential future re-use), is configured for length-1 lookahead, so the iterator it exposes produces 1-element arrays containing the currently read token (p2/com/tiger/scanner/TigerToken.java) from the file being read.

The LL(1) parser, located in p2/com/tiger/parser/TigerParser.java, makes use of the LL(1) auto-generated parse table and the production list in p2/com/tiger/parser/LLTable.java and the scanner's iterator to construct the initial AST. The stack used in the algorithm contains instances of the class in p2/com/tiger/syntax/TigerSymbol.java, which can be a TokenType (p2/com/tiger/scanner/TokenType.java, a terminal symbol template), or a TigerNT (p2/com/tiger/parser/TigerNT.java, a nonterminal).

The AST is isolated to p2/com/tiger/syntax/TigerAST.java. It maintains a private data structure, a tree of Node, NTNode and Tnode instances, which represent the actual syntax tree. This structure for a tree is the most comfortable and straightforward to maintain. TigerAST exposes methods to add terminals and non-terminals to it. TigerAST also exposes methods to revert the effects of left-factoring and left-recursion elimination, making the AST match the original grammar.

Automatic Table Generation:

The parse tables, including the LL parse table, have been automatically generated by the script parse_tiger.py in p2/com/tiger/parser/python. This script parses the grammar in tiger.txt, converts it to an LL(1) grammar by eliminating left recursion and left factoring, and records it in finaltiger.txt. It then generates the FIRST and FOLLOW sets, from them the FIRST+ set, and from it - the LL(1) parse table, which is put into parse_table.csv. It then generates the Java file p2/com/tiger/parser/LLTable.java (therefore this file should not be modified). This file contains a list of productions (the class in p2/com/tiger/parser/TigerProduction.java) as well as the LL(1) parse table, which can be indexed by a non-terminal (Top-Of-Stack from the LL(1) parse algorithm) and a terminal (token). The nonterminals added by left factoring and left recursion removal are labeled with the appropriate values of the NTType (p2/com/tiger/parser/NTType.java) enum.

Key Design Decisions:

In TigerParser, when the TOS (Top-of-Stack) is a terminal, the parsed word from the iterator is added to the AST, and for the nonterminal, it is just added to the AST itself.

In TigerAST, children are added as children to the lowest leftmost nonterminal which doesn't have all of the children defined in its respective production set (the number of children in the respective production is passed to the addSymbol method along with the non-terminal; this works because the LIFO addition order guarantees we need no other checks).

The effects of left factoring are reverted by replacing, in each node, an LF-type child with its own children. The effects of left recursion are reverted by treating the rightmost children of LR-nodes as a linked list of sorts and using this to reverse it, re-linking the nodes in a reverse order with their leftmost children and setting their NTType type to NORMAL.

Challenges and Issues:

The biggest challenge was finding a bug caused by the grammar having the same name for the “type” terminal used in typedecls and the “type” terminal. We solved it by adding a special case to the Python table generator script.

Testing:

The parser has been tested against the test cases provided on GitHub. We deemed them satisfactory as they cover all of the crucial edge cases.

Unresolved Issues:

None that we could find.