Whitney Henderson
Matthew Compton
Ben Bariteau
Nathan Fan

CS 3240 Project Part 2 Report

LL(1) Parser Generator

Takes in the grammar and the Token Spec and outputs accept or an error. It first parses the token spec file and builds a DFA for the token spec, then uses the token spec file to parse the grammar file. It parses the grammar file, and the grammar objects create parse tables. This part of the project is very similar to part one, and is in fact married to it, but this parser generator parses tokens as opposed to individual characters. We initially approached the problem by parsing individual characters, but under those constraints the grammar was ambiguous (and therefore not LL(1)). By parsing with tokens our grammar is no longer ambiguous and therefore creates no shift or reduce errors.

LL(1) Parsing Table

The parse table is contained in the Parse Table class. It creates a parse tree from the input symbol lists and rules, and returns the tree to be made into an NFA, which is then converted to a DFA, which is used as our parse table.

LL(1) Driver

This is used to parse the input script. This is also contained in the Parse Table class. It parses the input script using the parse table and the token spec DFA. It parses the input into tokens and the grammar into tokens, and uses the DFA to parse the tokens. Naturally it also uses a parsing stack, which we print to show how our driver is functioning, as opposed to merely stating accept or reject at the end of a run.

Error Marking

If a portion of the input is rejected by the parse table, then there is an error, and the error and its location are noted and printed. If there are no errors, an accept message is printed.

Assumptions

- We assumed that the tokens will follow these specs: \$([A-Z]+[A-Z-_]+[A-Z]+|[A-Z]+)
- Grammar can be completely converted into rules and tokens - essentially, the grammar must be capable of being parsed by the token spec.
- Every string that is in the grammar, that is not a rule, must be a string recognized by a token or a token itself.
- Whenever a token is directly referenced in the grammar, it is assumed to start with a capital letter (and no other things can be).
- Precedence to token recognition is given by the order of appearance - those which are

seen earlier are given precedence, which is an implicit assumption of importance based solely on occurrence.
- We assume that we don't need to include how an LL(1) parser works in the report.

Problems

- Default Identifier runs out of memory when converting from a DFA to a NFA. This is a problem which only appeared on occasion. We have run the code where this error has not occurred, and at times it has occurred. It is somewhat beyond our scope to fix.
- The grammar has a statement rule with three different productions. Initially we attempted to parse it as an LL(1) grammar character by character, but the issue is that ID is very obviously not LL(1) because it is ambiguous. ID can capture any sequence between one and ten characters. Rather than parsing as characters though, it is important to parse as tokens. Approaching the grammar this way makes it no longer ambiguous.
- All units were completed. Under our testing constraints, the code worked in it's entirety and we met all requirements.

Test Cases

1. The original testcase given by the TA's
2. A modified version of the original testcase - shorter length
3. A modified version of the original testcase - longer length
4. A modified version of the original testcase - whitespace heavy case
5. A modified version of the original testcase - empty/null case