

A2 Design

This document lists each non-terminal in the source grammar with an explanation of how it was implemented in csc488.cup.

program

Taken from the source grammar.

statement

Implementing a sequence of statements as `statement statement` produces shift/reduce conflicts due to ambiguity. We avoid this ambiguity by adding a non-terminal called `statement_redir` that produces all possible statements except for a sequence of statements. The non-terminal `statement` then produces `statement_redir` for a single statement, or `statement_redir statement` for a sequence of statements.

To prevent a sequence of unscoped statements from being in if-statements or while-loops, we switch out `statement` in the source grammar for `statement_redir`.

Embedded if-statements can create “dangling else” ambiguity so we use the rule `precedence nonassoc ELSE;` to make sure `ELSE` binds to the nearest `IF`.

declaration

Implementing a sequence of declarations as `declaration declaration` produces shift/reduce conflicts due to ambiguity. We avoid this ambiguity by adding a non-terminal called `declaration_redir` that produces all possible declarations except for a sequence of declarations. The non-terminal `declaration` then produces `declaration_redir` for a single declaration, or `declaration_redir declaration` for a sequence of declarations.

variablenames

Implementing a sequence of variablenames as `variablenames variablenames` produces shift/reduce conflicts due to ambiguity. We avoid this ambiguity by adding a non-terminal called `variablenames_redir` that produces all possible ways to form a variable name. The non-terminal `variablenames` then produces `variablenames_redir` for a single variable name, or `variablenames_redir variablenames` for a sequence.

bound

Taken from the source grammar.

generalBound

Taken from the source grammar.

scope

Taken from the source grammar.

output

Implementing an output sequence as `output COMMA output` produces shift/reduce conflicts due to ambiguity. We avoid this ambiguity by adding a non-terminal called `output_redir` that produces all possible outputs except for an output sequence. The non-terminal `output` then produces `output_redir` for a single output, or `output_redir COMMA output` for an output sequence.

input

Implementing an input sequence as `input COMMA input` produces shift/reduce conflicts due to ambiguity. We are able to remove ambiguity by implementing the sequence as `variable COMMA input`.

type

Taken from the source grammar.

arguments

Implementing an actual parameter sequence as `arguments COMMA arguments` produces shift/reduce conflicts due to ambiguity. We are able to remove ambiguity by implementing the sequence as `expression COMMA arguments`.

parameters

Implementing a formal parameter sequence as `parameters COMMA parameters` produces shift/reduce conflicts due to ambiguity. We are able to remove ambiguity by implementing the sequence as `IDENT COLON type COMMA parameters`.

variable

Taken from the source grammar.

expression

To ensure the correct precedence of operators, we decompose the expression into 7 different productions – `e0`, `e1`, `e2`, `e3`, `e4`, `e5`, `e6`, and `expression` – to separate the operators of different levels. Before any operations can take place, an expression is reduced to `e0`, and then is passed on to the `e1` rule, which contains the first level of operations. The result gets passed to the next rule for the next level of operations, until the whole thing is reduced to an `expression`.

variablename

Implemented directly as `IDENT` in `variablenames` and `variable` to avoid reduce/reduce conflicts with other non-terminals that produce only an identifier in the source grammar.

arrayname

Implemented directly as `IDENT` in `variable` to avoid reduce/reduce conflicts with other non-terminals that produce only an identifier in the source grammar.

functionname

Implemented directly as `IDENT` in `declaration`, and for function calls with arguments in `e0` to avoid reduce/reduce conflicts with other non-terminals that produce only an identifier in the source grammar. When a function with no arguments is called in `e0`, the production to `variable` suffices because that can also be reduced from `IDENT`.

parametername

Implemented directly as `IDENT` in `parameters` to avoid reduce/reduce conflicts with other non-terminals that produce only an identifier in the source grammar.

procedurename

Implemented directly as `IDENT` in `declaration` to avoid reduce/reduce conflicts with other non-terminals that produce only an identifier in the source grammar.