CS 347
Assignment 2
Compilers Lab

Anup Agarwal (150101009)
Chandan Satti (150101059)
Surabhi Gupta (150101078)

Department Of Computer Science And Engineering
IIT Guwahati
12 March 2018

Introduction:

Language Supports:

- Iteration statements (for, while, do-while)
- conditional statements (if else)
- nested loops, expressions and conditionals
- arrays
- Type Specifiers : {void, bool, int, float, char, Processor, Link, Cluster, Job, Memory, **Scheduler**}

- Operators:
  - Unary operators (*, +, -, ~, !)
  - Arithmetic (+, -, *, /, %)
  - Relational (<=, >=, >, <, ==, !=)
  - Bitwise Operators (&, |, ^)
  - Logical Operators (&&, ||)
  - Conditional Operator (? :)

Basic Production Rules:

```
statement_list: statement statement_list
| /* EPSILON */

statement: var_decl;
| expression_statement
| selection_statement
| iteration_statement
| ‘{‘ statement_list ‘}’
| declaration
```

The program is essentially a statement_list. Each statement can be either a variable declaration, expression, selection , iteration or a declaration.

statement -> ‘{‘ statement_list ’}’ allows provision for nesting and compound statements.

```
iteration_statement
: WHILE OPEN_BRACKET expression CLOSED_BRACKET statement
| DO statement WHILE OPEN_BRACKET expression CLOSED_BRACKET SEMI
| FOR OPEN_BRACKET expression_statement expression_statement CLOSED_BRACKET statement
| FOR OPEN_BRACKET expression_statement expression_statement expression CLOSED_BRACKET
statement
| FOR OPEN_BRACKET declaration expression_statement CLOSED_BRACKET statement
| FOR OPEN_BRACKET declaration expression_statement expression CLOSED_BRACKET
statement

selection_statement
: IF OPEN_BRACKET expression CLOSED_BRACKET statement ELSE statement
| IF OPEN_BRACKET expression CLOSED_BRACKET statement
```

Iteration_statements deals with loops in which we have done the while, do while and the for loop. Multiple declarations for the FOR loops are cases in which FOR() has 2 or 3 arguments and the first argument can also be a declaration. (2nd argument is the conditional and third is the iteration)
Also there are the selection productions i.e. the if else statements.

```
declaration: type_specifier declarator ASSIGN assignment_expression
SEMI

declarator: ID
     | declarator OPEN_SQUARE CLOSED_SQUARE
```

These productions provide for declarations for variables and arrays.
type_specifier takes one of the types described in the introduction slide.

Operator Precedence Explanation:

For each precedence level a non terminal is made:

- Operators in precedence:
  - Unary operators (*, +, -, ~, !) : unary_expression
  - Multiplicative (*, /, %) : multiplicative_expression
  - Additive (+, -) : additive_expression
  - Relational (<=, >=, >, <) : relational_expression
  - Equality (==, !=) : equality_expression
  - Bitwise Operators (&, ^, |) : {and, exclusive_or, inclusive_or}_expression
  - Logical Operators (&&, ||) : logical_{and, or}_expression
  - Conditional Operator (? :) : conditional_expression

Operator Precedence Explanation:

For each non terminal productions are added as:

```
lower
: higher | lower OPERATOR higher
```

For e.g:

```
additive_expression
: multiplicative_expression
| additive_expression PLUS multiplicative_expression
| additive_expression MINUS multiplicative_expression
```

Some other important productions:

```
postfix_expression
: postfix_expression OPEN_SQUARE expression CLOSED_SQUARE (array accesses)
| postfix_expression OPEN_BRACKET argument_expression_list CLOSED_BRACKET
(function calls)
| postfix_expression DOT ID (class data member access)
| ID DOT mem_func (inbuilt member function call)
| postfix_expression PTR_OP ID (pointer dereferencing)
| OPEN_CURLY initializer_list CLOSED_CURLY
| OPEN_SQUARE initializer_list CLOSED_SQUARE
;

initializer
: OPEN_CURLY initializer_list CLOSED_CURLY
| OPEN_SQUARE initializer_list CLOSED_SQUARE
| assignment_expression
;
```

```
initializer_list
: initializer
| initializer_list COMMA
initializer
;

argument_expression_list
: assignment_expression
| argument_expression_list
COMMA assignment_expression
;
```

Overview of the Classes (Jobs, Links…)

Each of the custom type specifier is interpreted as a class, having a constructor, member functions and parameters.

The constructor support optional parameters as well as parameters passed by referencing the name of the formal parameter (i.e. `func(name = “xyz”))`

The constructor is of the form:

```
constructor: PROCESSOR OPEN_BRACKET param_list_proc
CLOSED_BRACKET
```

NOTE: ‘primary_expression’ derives ‘constructor’ which allows for in place declarations in the function calls and constructors.

Implementing Optional Parameters

```
param_list_cluster : cluster_param1 COMMA
cluster_param2 COMMA cluster_param3 COMMA
cluster_param4 opt_cluster_param5 opt_cluster_param6

cluster_param1: assignment_expression

cluster_param2: TOPOLOGY assign_colon
assignment_expression
| assignment_expression

cluster_param3
: LINK_BANDWIDTH assign_colon assignment_expression
| assignment_expression
```

Have a non terminal for each parameter, the ones which are optional have EPSILON productions and the ones before optional parameters have optional COMMAS

NOTE: all type checks will be done at time of Semantic Analysis

```
cluster_param4: LINK_CAPACITY assign_colon
assignment_expression
| assignment_expression
| LINK_CAPACITY assign_colon
assignment_expression COMMA
| assignment_expression COMMA

opt_cluster_param5
: SCHED assign_colon assignment_expression
| assignment_expression
| SCHED assign_colon assignment_expression
COMMA
| assignment_expression COMMA
| /* EPSILON */

opt_cluster_param6
: NAME assign_colon assignment_expression
| /* EPSILON */
```

Parsing for Scheduler

The schedulers are of 2 types: distributed schedulers, individual processor scheduler.

User defines an object of class Scheduler as:

```
Scheduler(algo = (string or function pointer), name = (string))
```

If algo is a string then it is a well known scheduler (eg: SJF, FCFS, RR; Monolithic, YARN, Mesos etc.)

Based on the string value or the prototype of the function the type of the scheduler is determined.
NOTE: The constructor is overloaded, viz.

```
Scheduler(string, void nextJobAction(proc_list, job_list))
Scheduler(string, void nextJobAction(cluster_list))
```

This scheduler is an optional parameter for the classes: Processor and Cluster.

When run will be invoked, the appropriate schedulers will be used.