Languages-beta: MiniJava-Syntax *

The PLanCompS Project

`MiniJava-Syntax.cbs` | PLAIN | PRETTY

Language "`MiniJava`"

[The MiniJava Reference Manual]: http://www.cambridge.org/us/features/052182060X/mjreference/mjreference.html

[Modern Compiler Implementation in Java: the MiniJava Project]: http://www.cambridge.org/us/features/052182060X/

The grammar used here is mostly a transliteration of the one provided at: http://www.cambridge.org/us/features/052182(
(which differs in trivial ways from the one in the cited reference manual).

The rest of this file gives an overview of the MiniJava syntax. It is mostly in the form of a comment with embedded productions. The nonterminal symbols are hyperlinks to their actual specifications; similarly, section numbers (such as **1** below) link to the corresponding specification section.

1 Programs

Syntax

START : start ::= program

P : program ::= main-class class-declaration*

MC : main-class ::= 'class' identifier '{'
'public' 'static' 'void' 'main' '(' 'String' '[' ']' identifier ')' '{'
statement
'}'
'}'

2 Declarations

*Suggestions for improvement: `plancomps@gmail.com`.
Reports of issues: `https://github.com/plancomps/CBS-beta/issues`.

Syntax *CD* : class-declaration ::= 'class' identifier ('extends' identifier)? '{'
var-declaration*
method-declaration*
'}'

VD : var-declaration ::= type identifier ';'

MD : method-declaration ::= 'public' type identifier '(' formal-list? ')' '{'
var-declaration*
statement*
'return' expression ';'
'}'

T : type ::= 'int' '[' ']'
| 'boolean'
| 'int'
| identifier

FL : formal-list ::= type identifier (',' formal-list)?

3 Statements

Syntax *S* : statement ::= '{' statement* '}'
| 'if' '(' expression ')' statement 'else' statement
| 'while' '(' expression ')' statement
| 'System' '.' 'out' '.' 'println' '(' expression ')' ';'
| identifier '=' expression ';'
| identifier '[' expression ']' '=' expression ';'

4 Expressions

Syntax *E* : expression ::= expression '&&' expression
| expression '<' expression
| expression '+' expression
| expression '-' expression
| expression '*' expression
| expression '[' expression ']'
| expression '.' 'length'
| expression '.' identifier '(' expression-list? ')'
| integer-literal
| 'true'
| 'false'
| identifier
| 'this'
| 'new' 'int' '[' expression ']'
| 'new' identifier '(' ')'
| '!' expression
| '(' expression ')'

EL : expression-list ::= expression (',' expression-list)?

5 Lexemes

Lexis

```
ID : identifier  ::=  letter (letter | digit | '_')*
IL : integer-literal  ::=  digit+
letter  ::=  'a'–'z' | 'A'–'Z'
digit  ::=  '0'–'9'
```

6 Disambiguation

The mixture of CBS and SDF below specifies how MiniJava texts are to be disambiguated by parsers generated from the above grammar.

The specified rules are adequate to disambiguate all the example programs provided at https://www.cambridge.org/us/feat

Syntax SDF

```
context-free syntax
  expression  ::=  expression '*' expression {left}
  expression  ::=  expression '+' expression {left}
  expression  ::=  expression '-' expression {left}
  expression  ::=  expression '<' expression {non-assoc}
  expression  ::=  expression '&&' expression {left}

context-free priorities
{
  expression  ::=  expression '.' identifier '(' expression-list? ')'
  expression  ::=  expression '[' expression ']'
} <0> >
  expression  ::=  '!' expression
>
  expression  ::=  expression '*' expression
> {
  expression  ::=  expression '+' expression
  expression  ::=  expression '-' expression
} >
  expression  ::=  expression '<' expression
>
  expression  ::=  expression '&&' expression
```

Lexis SDF

```
lexical restrictions
identifier -/- [a-zA-Z0-9_]
integer-literal -/- [0-9]

lexical syntax
identifier = reserved-id {reject}
```

Lexis reserved-id ::= `'String'`
| `'System'`
| `'boolean'`
| `'class'`
| `'else'`
| `'extends'`
| `'false'`
| `'if'`
| `'int'`
| `'length'`
| `'main'`
| `'new'`
| `'out'`
| `'println'`
| `'public'`
| `'return'`
| `'static'`
| `'this'`
| `'true'`
| `'void'`