Languages-beta: MiniJava-Syntax

The PLanCompS Project

`Languages-beta/MiniJava/MiniJava-Syntax/MiniJava-Syntax.cbs`*

Language "`MiniJava`"

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

[Modern Compiler Implementation in Java: the MiniJava Project]: http://www.cambridge.org/us/features/052

The grammar used here is mostly a transliteration of the one provided at: http://www.cambridge.org/us/features/052182060X/grammar.html (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 }`

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

2 Declarations

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`

T : type ::= `int [ ]`
| `boolean`
| `int`
| identifier

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

3 Statements

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

4 Expressions

$$
\begin{aligned}
E : \text{expression} \;::=\; & \text{expression}\ \texttt{\&\&}\ \text{expression} \\
\mid\; & \text{expression}\ \texttt{<}\ \text{expression} \\
\mid\; & \text{expression}\ \texttt{+}\ \text{expression} \\
\mid\; & \text{expression}\ \texttt{-}\ \text{expression} \\
\mid\; & \text{expression}\ \texttt{*}\ \text{expression} \\
\mid\; & \text{expression}\ \texttt{[}\ \text{expression}\ \texttt{]} \\
\mid\; & \text{expression}\ \texttt{.}\ \texttt{length} \\
\mid\; & \text{expression}\ \texttt{.}\ \text{identifier}\ \texttt{(}\ \text{expression-list}^{?}\ \texttt{)} \\
\mid\; & \text{integer-literal} \\
\mid\; & \texttt{true} \\
\mid\; & \texttt{false} \\
\mid\; & \text{identifier} \\
\mid\; & \texttt{this} \\
\mid\; & \texttt{new int [}\ \text{expression}\ \texttt{]} \\
\mid\; & \texttt{new}\ \text{identifier}\ \texttt{(}\ \texttt{)} \\
\mid\; & \texttt{!}\ \text{expression} \\
\mid\; & \texttt{(}\ \text{expression}\ \texttt{)} \\
EL : \text{expression-list} \;::=\; & \text{expression}\ (\texttt{,}\ \text{expression-list})^{?}
\end{aligned}
$$

5 Lexemes

Lexis

$$
\begin{aligned}
ID : \text{identifier} \;::=\; & \text{letter}\ (\text{letter} \mid \text{digit} \mid \texttt{\_})^{*} \\
IL : \text{integer-literal} \;::=\; & \text{digit}^{+} \\
\text{letter} \;::=\; & \texttt{a} - \texttt{z} \\
\mid\; & \texttt{A} - \texttt{Z} \\
\text{digit} \;::=\; & \texttt{0} - \texttt{9}
\end{aligned}
$$

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 http://www.cambridge.org/us/features/052182060X/#progs

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`