MiniJava Language Specification

Daniel Grund*

October 18, 2011

1 Introduction

MiniJava is a subset of the programming language Java. Hence, it is possible to compile MiniJava programs to byte code with a regular Java compiler. MiniJava features many concepts that are interesting for compiler construction. However, it is restricted such that its implementation is feasible within the context of a lecture: The language contains only few statements as well as expressions and requires only a simple run-time system. It abstains from many features of Java that only complicate the run-time system and translation unnecessarily, e.g. exceptions and multi-threading.

2 Properties

2.1 Type system

MiniJava knows the base types `int` for integers and `boolean` for logical values. User-defined types are classes and method types. Classes contain attributes and methods. Methods have a method type, which indicates the number and type of the method's parameters as well as the method's return type. Method overloading (several methods with the same name but different parameter and/or return types) is not allowed. On method invocation, the number and type of arguments must match the number and type of the method's parameters.

2.2 Run-time System

MiniJava only has a minimal run-time system without large standard libraries. Execution of a program starts at the main method, which must always be present. There is no automatic memory management, i.e. MiniJava programs can only allocate memory; memory cannot be freed for reuse.

*Thanks to all prior teaching assistants and anybody who contributed to the MiniJava project.

3 Lexical Elements

MiniJava has the following lexical elements:

- White space: These are space, new line, carriage return and tabulator.
- Comments: The string `/*` followed by any characters until the terminating `*/` is a comment. Comments are not nestable, further `/*` within a comment are ignored;[1] a comment always ends when the first `*/` is encountered.
- Keywords and operators: All tokens printed in `bold` in the grammar specification are keywords or operators.
- `IDENT`: An identifier starts with an underscore or letter and is followed by any number of letters, underscores and digits. Only the letters A to Z and a to z are allowed, case is important. Keywords are not `IDENT`s.
- `INTEGER_LITERAL`: A decimal integer literal is a digit sequence starting with any of the digits 1 to 9 and is followed by any number of digits 0 to 9. A single 0 is also an integer literal.

Comments and white space have no meaning except for separating tokens.

4 Syntax

- Non-terminals are printed in italic. Example: *Expression*.
- Terminals are printed in bold typewriter font. Example: `public`.
- X *?* means no or exactly one occurrence of X.
- X * means any number of occurrences of X (in particular no occurrence).
- | indicates alternatives.
- *( )* is for grouping multiple syntactical elements.

[1] The most elegant solution is to issue a warning if a `/*` is encountered within a comment.

Program → *ClassDeclaration**

ClassDeclaration → `class IDENT {` *ClassMember** `}`

ClassMember → *Field* | *Method* | *MainMethod*

Field → `public` *Type* `IDENT ;`

MainMethod → `public static void main ( String [ ] IDENT )` *Block*

Method → `public` *Type* `IDENT (` *Parameters?* `)` *Block*

Parameters → *Parameter* | *Parameter* `,` *Parameters*

Parameter → *Type* `IDENT`

Type → *Type* `[ ]` | `int` | `boolean` | `void` | `IDENT`

Statement → *Block*
| *EmptyStatement*
| *IfStatement*
| *PrintStatement*
| *ExpressionStatement*
| *WhileStatement*
| *ReturnStatement*

Block → `{` *BlockStatement** `}`

BlockStatement → *Statement* | *LocalVariableDeclarationStatement*

LocalVariableDeclarationStatement → *Type* `IDENT`*(*`=` *Expression)?* `;`

EmptyStatement → `;`

WhileStatement → `while (` *Expression* `)` *Statement*

IfStatement → `if (` *Expression* `)` *Statement (*`else` *Statement)?*

PrintStatement → `System . out . println (` *Expression* `) ;`

ExpressionStatement → *Expression* `;`

ReturnStatement → `return` *Expression?* `;`

Expression → *AssignmentExpression*

AssignmentExpression → *LogicalOrExpression (*`=` *AssignmentExpression)?*

LogicalOrExpression → *(LogicalOrExpression* `||`*)? LogicalAndExpression*

LogicalAndExpression → *(LogicalAndExpression* `&&`*)? EqualityExpression*

EqualityExpression → *(EqualityExpression (*`==` | `!=`*))? RelationalExpression*

RelationalExpression → *(RelationalExpression (*`<` | `<=` | `>` | `>=`*))? AdditiveExpression*

AdditiveExpression → *(AdditiveExpression (*`+` | `-`*))? MultiplicativeExpression*

MultiplicativeExpression → *(MultiplicativeExpression (*`*` | `/` | `%`*))? UnaryExpression*

UnaryExpression → *PrimaryExpression* | *(*`!` | `-`*) UnaryExpression*

PrimaryExpression → `null`
| `false`
| `true`
| `INTEGER_LITERAL`
| *MethodInvocationExpression*
| *FieldAccessExpression*
| *LocalVariableReferenceExpression*
| `this`
| `(` *Expression* `)`
| *NewObjectExpression*
| *NewArrayExpression*

MethodInvocationExpression → *(PrimaryExpression* `.`*)?* `IDENT` `(` *ExpressionList?* `)`
ExpressionList → *Expression(*`,` *Expression)**
FieldAccessExpression → *(PrimaryExpression* `.`*)?* `IDENT`
LocalVariableReferenceExpression → `IDENT`
NewObjectExpression → `new` *Type* `(` `)`
NewArrayExpression → `new` *Type* `[` *Expression* `]`*(*`[` `]`*)**

5 Semantics

Except for a few exceptions the semantics of MiniJava is consistent with the semantics of Java. The latter is described in [GJSB00]. The exceptions are:

- The method main(String[] args) must not be called.
- The definite assignment rules of Java need not be checked.

References

[GJSB00] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java Language Specification, Second Edition: The Java Series*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.