MiniJava-Sprachbericht Version 3.2

Matthias Braun Jürgen Graf

17. Oktober 2012

1 Einleitung

MiniJava ist eine Untermenge der Programmiersprache Java. Daher können Programme in MiniJava von jedem Java-Compiler zu Bytecode übersetzt werden. Die Sprache enthält viele für den Übersetzerbau interessante Konzepte wie rekursive Methodenaufrufe. Trotzdem erlaubt ihre Kompaktheit eine Implementierung im Rahmen eines universitären Praktikums. In MiniJava wird auf viele Eigenschaften von Java, die das Laufzeitsystem und die Übersetzung unnötig erschweren, wie z.B. Ausnahmen und Mehrfädigkeit, verzichtet.

MiniJava ist, wie Java, eine objektorientierte Sprache. Sie verfügt jedoch nur über wenige Anweisungen sowie wenige Ausdrücken und benötigt nur ein sehr einfaches Laufzeitsystem.

2 Eigenschaften

2.1 Typsystem

MiniJava kennt als Basistyp Wahrheitswerte `boolean` und Ganzzahlen `int`. `int`-Werte sind als vorzeichenbehaftete 32Bit-Zahlen im 2er-Komplement kodiert. Der Typ `boolean` umfasst die Werte `true` und `false`. Benutzerdefinierte Typen sind Klassen, Reihungen und Methodentypen. Klassen enthalten Felder und Methoden. Reihungen können auf Basis- und benutzerdefinierten Typen basieren. Methoden besitzen einen Methodentyp, der die Anzahl und Typen der Parameter sowie den Typ des Rückgabewerts, falls die Methode Werte zurückgibt, angibt. Werden keine Werte zurückgeliefert, so wird `void` als Platzhalter für den Typ des Rückgabewerts verwendet. Methoden mit gleichem Bezeichner aber verschiedenen Parameter-/Ergebnistypen sind nicht erlaubt (keine polymorphen Aufrufe). Eine Methode und ein Feld mit gleichem Namen in der gleichen Klasse ist allerdings erlaubt. Beim Aufruf von Methoden müssen die Typen der Argumente denen der Parameter entsprechen.

2.2 Laufzeitsystem

MiniJava besitzt ein minimalistisches Laufzeitsystem ohne große Standardbibliotheken. Die Ausführung eines Programms beginnt an der immer vorhandenen **`main`**-Methode. Es gibt keine automatische Speicherverwaltung. MiniJava Programme können zwar Speicher anfordern, dieser kann aber weder manuell freigegeben werden noch werden nicht mehr benötigte Objekte automatisch entfernt. Die Standardbibliothek enthält als eine einzige Funktion `System.out.println`, die eine Ganzzahl gefolgt von einem Zeilenumbruch ausgibt.

3 Übersetzung

MiniJava-Programme bestehen aus einer Eingabedatei die das komplette zu übersetzende Programm enthält. Die Eingabe ist im ASCII-Zeichensatz kodiert.

3.1 Wortschatz

MiniJava besteht aus den folgenden Wörtern:

- **`Leerraum`**: Dazu zählen Leerzeichen, Zeilenumbruch, Wagenrücklauf und Tabulator.
- **`Kommentar`**: Kommentare werden durch die Zeichenkette `/*` eingeleitet. Ein Kommentar endet beim nächsten Auftreten der Zeichenkette `*/`. Ein fehlendes `*/` ist ein Fehler. Daraus ergibt sich, dass Kommentare nicht geschachtelt werden können, weitere `/*` innerhalb eines Kommentars werden ignoriert;[1] beim ersten `*/` ist der Kommentar stets zu Ende.
- Schlüsselwörter sind:

  `abstract`, **`assert`**, **`boolean`**, **`break`**, **`byte`**, **`case`**, **`catch`**, **`char`**, **`class`**, **`const`**, **`continue`**, **`default`**, **`double`**, **`do`**, **`else`**, **`enum`**, **`extends`**, **`false`**, **`finally`**, **`final`**, **`float`**, **`for`**, **`goto`**, **`if`**, **`implements`**, **`import`**, **`instanceof`**, **`interface`**, **`int`**, **`long`**, **`native`**, **`new`**, **`null`**, **`package`**, **`private`**, **`protected`**, **`public`**, **`return`**, **`short`**, **`static`**, **`strictfp`**, **`super`**, **`switch`**, **`synchronized`**, **`this`**, **`throws`**, **`throw`**, **`transient`**, **`true`**, **`try`**, **`void`**, **`volatile`**, **`while`**

- Operatoren und Separatoren sind: `!=`, `!`, `(`, `)`, `*=`, `*`, `++`, `+=`, `+`, `,`, `-=`, `--`, `-`, `.`, `/=`, `/`, `:`, `;`, `<<=`, `<<`, `<=`, `<`, `==`, `=`, `>=`, `>>=`, `>>>=`, `>>>`, `>>`, `>`, `?`, `%=`, `%`, `&=`, `&&`, `&`, `[`, `]`, `^=`, `^`, `{`, `}`, `~`, `|=`, `||`, `|`
- **`IDENT`**: Ein Bezeichner, er beginnt mit einem Buchstaben oder Unterstrich gefolgt von beliebig vielen Buchstaben, Unterstrichen oder Ziffern in beliebiger Reihenfolge. Schlüsselwörter sind keine **`IDENT`**s.

[1] Die eleganteste Lösung ist bei `/*` innerhalb des Kommentars eine Warnung auszugeben.

- `INTEGER_LITERAL`: Dezimale Zahlenliterale beginnen mit einer der Ziffern von `1` bis `9`, gefolgt von beliebig vielen weiteren Ziffern `0` bis `9`. Eine einzelne `0` ist ebenso eine Zahlenliteral.

Kommentare und Leerzeichen haben keine Bedeutung. Ihre Funktion ist die Trennung von Wörtern, Dokumentation des Quelltexts und sie sorgen für ästhetische Programme. Sie werden für syntaktische Belange ignoriert.

3.2 Syntax

- Nichtterminale sind kursiv gedruckt. Beispiel: *Expression*.
- Terminale fett in Schreibmaschinenschrift. Beispiel: **`public`**.
- *X?* bedeutet kein oder genau ein Vorkommen von *X*.
- *X** bedeutet beliebig viele Vorkommen von *X* (insbesondere auch kein Vorkommen).
- *( )* wird zum Klammern von syntaktischen Gruppen benutzt.

Program → *ClassDeclaration**

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

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

Field → **`public`** *Type* **`IDENT ;`**

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

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

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

Parameter → *Type* **`IDENT`**

Type → *Type* **`[ ]`** | *BasicType*

BasicType → **`int`** | **`boolean`** | **`void`** | **`IDENT`**

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

Block → **`{`** *BlockStatement** **`}`**

BlockStatement → *Statement* | *LocalVariableDeclarationStatement*

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

EmptyStatement → **;**

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

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

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 → *PostfixExpression* | *(***!** | **-***) UnaryExpression*

PostfixExpression → *PrimaryExpression (PostfixOp)**

PostfixOp → *MethodInvocation*
| *FieldAccess*
| *ArrayAccess*

MethodInvocation → **. IDENT (** *Arguments* **)**

FieldAccess → **. IDENT**

ArrayAccess → **[** *Expression* **]**

Arguments → *(Expression (***,** *Expression)*)?*

PrimaryExpression → **null**
| **false**
| **true**
| **INTEGER_LITERAL**
| **IDENT**
| **IDENT (** *Arguments* **)**
| **this**
| **(** *Expression* **)**
| *NewObjectExpression*
| *NewArrayExpression*

NewObjectExpression → **new IDENT ( )**

NewArrayExpression → **new** *BasicType* **[** *Expression* **]** *(***[ ]***)**

4 Semantik

Die Semantik von MiniJava entspricht bis auf kleine Ausnahmen der Semantik der Sprache Java. Diese ist in [GJSB00] beschrieben.

Ausnahmen:

- In einem Programm darf nur eine einzige **public static** Methode mit Namen **main** existieren.
- Die main(String[] args) -Methode darf nicht aufgerufen werden. Der Parameter der Methode existiert zwar, ein Zugriff darauf ist allerdings nicht erlaubt.
- Der Ausdruck **System.out.println(***Expression***)** ist ein Sonderfall: Ist kein Bezeichner **System** bekannt, so erzeugt dieser Ausdruck einen Aufruf der **System.out.println** Funktion in der Standardbibliothek.
- Die „Definite Assignment"-Regeln von Java müssen nicht beachtet werden. Werden uninitialisierte lokale Variablen benutzt, so ist das Verhalten nicht spezifiziert. Felder von Objekten sind, wie von Java gewohnt, mit 0, **false** bzw. **null** initialisiert.
- Ausnahmen treten nicht auf. Das Verhalten in Situationen wie dem Zugriff auf **null**-Referenzen oder Division durch 0 ist nicht spezifiziert.
- Es gibt keinen Vererbungsmechanismus und es existiert keine Klasse, die allen anderen zugrunde liegt (also kein `System.lang.Object`).
- Arrays sind keine normalen Objekte. Man kann nur mit **[]**-Ausdrücken auf sie zugreifen. Sie beinhalten insbesondere keine Felder wie **array.length**.
- Das Verhalten beim Zugriff auf eine Reihung außerhalb deren Grenzen ist nicht spezifiziert.

Literatur

[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.