The MiniJava Type System

Jens Palsberg

October 2014

1 What is MiniJava?

MiniJava is a subset of Java. The meaning of a MiniJava program is given by its meaning as a Java program. Overloading is not allowed in MiniJava. The MiniJava statement `System.out.println(...);` can only print integers. The MiniJava expression `e.length` only applies to expressions of type `int[]`.

We will now specify MiniJava's syntax and type system. A MiniJava program will type check with the MiniJava type system (as specified below) if and only if it will type check with the Java type system (as specified in the Java Language Specification).

2 Syntax

The grammar below uses the following metanotation:

- Nonterminal symbols are words written in *this font*.
- Terminal symbols are written in `this font`, except ⟨IDENTIFIER⟩ and ⟨INTEGER_LITERAL⟩.
- A production is of the form *lhs* ::= *rhs*, where *lhs* is a nonterminal symbol and *rhs* is a sequence of nonterminal and terminal symbols, with choices separated by |, and some times using "..." to denote a possibly empty list.
- We will use superscripts and subscripts to distinguish metavariables.

$$
\begin{array}{rrcl}
(\textit{Goal}) & g & ::= & mc\ d_1\ \ldots\ d_n \\
(\textit{MainClass}) & mc & ::= & \texttt{class}\ id\ \{\ \texttt{public static void main (String [] }id^S\texttt{)}\{ \\
& & & \quad t_1\ id_1;\ \ldots;\ t_r\ id_r;\ s_1 \ldots\ s_q\}\} \\
(\textit{TypeDeclaration}) & d & ::= & \texttt{class}\ id\ \{\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \} \\
& & | & \texttt{class}\ id\ \texttt{extends}\ id^P\ \{\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \} \\
(\textit{MethodDeclaration}) & m & ::= & \texttt{public}\ t\ id^M\ \texttt{(}t_1^F\ id_1^F\texttt{,}\ \ldots\texttt{,}\ t_n^F\ id_n^F\texttt{)}\ \{ \\
& & & \quad t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_r\ id_r\texttt{;}\ s_1\ \ldots\ s_q\ \texttt{return}\ e\texttt{;}\ \} \\
(\textit{Type}) & t & ::= & \texttt{int[]}\ |\ \texttt{boolean}\ |\ \texttt{int}\ |\ id \\
(\textit{Statement}) & s & ::= & \{\ s_1\ \ldots\ s_q\ \}\ |\ id\ \texttt{=}\ e\texttt{;}\ |\ id\ \texttt{[}\ e_1\ \texttt{]}\ \texttt{=}\ e_2\texttt{;} \\
& & | & \texttt{if (}\ e\ \texttt{)}\ s_1\ \texttt{else}\ s_2\ |\ \texttt{while (}\ e\ \texttt{)}\ s\ |\ \texttt{System.out.println(}\ e\ \texttt{);} \\
(\textit{Expression}) & e & ::= & p_1\ \texttt{\&\&}\ p_2\ |\ p_1\ \texttt{<}\ p_2\ |\ p_1\ \texttt{+}\ p_2\ |\ p_1\ \texttt{-}\ p_2\ |\ p_1\ \texttt{*}\ p_2\ |\ p_1\ \texttt{[}\ p_2\ \texttt{]} \\
& & | & p\ \texttt{.length}\ |\ p\ \texttt{.}id\ \texttt{(}e_1\texttt{,}\ \ldots\texttt{,}\ e_n\texttt{)}\ |\ p \\
(\textit{PrimaryExpression}) & p & ::= & c\ |\ \texttt{true}\ |\ \texttt{false}\ |\ id\ |\ \texttt{this}\ |\ \texttt{new int[}e\texttt{]}\ |\ \texttt{new}\ id\texttt{()}\ |\ \texttt{!}e\ |\ \texttt{(}e\texttt{)} \\
(\textit{IntegerLiteral}) & c & ::= & \langle\textsf{INTEGER\_LITERAL}\rangle \\
(\textit{Identifier}) & id & ::= & \langle\textsf{IDENTIFIER}\rangle
\end{array}
$$

3 Notation for Rules

We will use the following notation:

$$\frac{\mathit{hypothesis}_1 \qquad \mathit{hypothesis}_2 \quad \dots \quad \mathit{hypothesis}_n}{\mathit{conclusion}}$$

This is a *rule* that says that if we can derive all of $\mathit{hypothesis}_1$, $\mathit{hypothesis}_2$, ..., $\mathit{hypothesis}_n$, then we can also derive *conclusion*.

A special case arises when $n = 0$: we can write this case as:

$$\frac{}{\mathit{conclusion}}$$

or we can even omit the horizontal bar and write:

$$\mathit{conclusion}$$

We can say that this case is a rule with no hypotheses, or we can call it an *axiom*.

A *derivation* happens when we begin with one or more axioms, then perhaps apply some rules, and finally arrive at a conclusion. Notice that we can organize a derivation as a tree that has the axioms as leaves and the conclusion as the root. We can refer to such as tree as a *derivation tree*.

4 Subtyping

We now define a *subtype* relation on class names. We use $\leq$ to denote the subtype relation. We $t_1 \leq t_2$, we say that t_1 is a subtype of t_2. We define that $\leq$ is reflexive and transitive, and generated by the "extends" relation among classes.

$$t \leq t \tag{1}$$

$$\frac{t_1 \leq t_2 \qquad t_2 \leq t_3}{t_1 \leq t_3} \tag{2}$$

$$\frac{\texttt{class}\ C\ \texttt{extends}\ D\ \texttt{...}\ \text{is in the program}}{C \leq D} \tag{3}$$

5 Type Environments

A type environment is a finite mapping from identifiers to types. We use A to range over type environments. We use $dom(A)$ to denote the domain of A. If $id_1, \dots, id_r$ are pairwise distinct identifiers, then the notation $[id_1 : t_1, \dots, id_r : t_r]$ denotes a type environment that maps id_i to t_i, for $i \in 1..r$. If A_1, A_2 are type environments, then $A_1 \cdot A_2$ is a type environment defined in the following way:

$$(A_1 \cdot A_2)(id) = \begin{cases} A_2(id) & \text{if } id \in dom(A_2) \\ A_1(id) & \text{otherwise} \end{cases} \tag{4}$$

Notice that A_2 takes precedence over A_1.

6 Helper Functions

6.1 The *classname* Helper Function

The function *classname* returns the name of a class. The definition of *classname* is:

$$classname(\texttt{class}\ id\ \texttt{\{ public static void main (String [] }\ id^S\texttt{)}\ \texttt{\{}\ \ldots\texttt{\}\}}) = id \tag{5}$$

$$classname(\texttt{class}\ id\ \texttt{\{}\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \texttt{\}}) = id \tag{6}$$

$$classname(\texttt{class}\ id\ \texttt{extends}\ id^P\ \texttt{\{}\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \texttt{\}}) = id \tag{7}$$

6.2 The *linkset* Helper Function

The function *linkset* returns the connection between a class and its superclass (represented as a singleton set with one pair of class names), or the emptyset if a class has no superclass. The definition of *linkset* is:

$$linkset(\texttt{class}\ id\ \texttt{\{ public static void main (String [] }\ id^S\texttt{)}\ \texttt{\{}\ \ldots\texttt{\}\}}) = \emptyset \tag{8}$$

$$linkset(\texttt{class}\ id\ \texttt{\{}\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \texttt{\}}) = \emptyset \tag{9}$$

$$linkset(\texttt{class}\ id\ \texttt{extends}\ id^P\ \texttt{\{}\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \texttt{\}}) = \{(id, id^P)\} \tag{10}$$

6.3 The *methodname* Helper Function

The function *methodname* returns the name of a method definition. The definition of *methodname* is:

$$methodname(\texttt{public}\ t\ id^M\ \texttt{(}\ldots\texttt{)}\ \ldots) = id^M \tag{11}$$

6.4 The *distinct* Helper Function

The *distinct* function checks that the identifiers in a list are pairwise distinct.

$$\frac{\forall i \in 1..n : \forall j \in 1..n : id_i = id_j \Rightarrow i = j}{distinct(id_1, \ldots, id_n)} \tag{12}$$

6.5 The *acyclic* Helper Function

The *acyclic* function checks that a set of pairs contains no cycles.

$$\frac{\neg\ [\ \exists j_1, \ldots, j_h \in 1..n : (\forall i \in 1..(h-1) : id^P_{j_i} = id_{j_{i+1}})\ \wedge\ (id^P_{j_h} = id_{j_1})\ ]}{acyclic(\{\ (id_1, id^P_1), \ldots, (id_n, id^P_n)\ \})} \tag{13}$$

6.6 The *fields* Helper Function

We use the notation $fields(C)$ to denote a type environment constructed from the fields of C and the fields of the superclasses of C. The fields in C take precedence over the fields in the superclasses of C. The definition of $fields$ is:

$$\frac{\texttt{class}\ id\ \texttt{\{}\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \texttt{\}}\quad \text{is in the program}}{fields(id) = [id_1 : t_1, \ldots, id_f : t_f]} \tag{14}$$

$$\frac{\texttt{class}\ id\ \texttt{extends}\ id^P\ \texttt{\{}\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \texttt{\}}\quad \text{is in the program}}{fields(id) = fields(id^P) \cdot [id_1 : t_1, \ldots, id_f : t_f]} \tag{15}$$

6.7 The *methodtype* Helper Function

We use the notation $methodtype(id, id^M)$ to denote the list of argument types of the method with name id^M in class id (or a superclass of id) together with the return type (or $\perp$ if no such method exists). The result of *methodtype* is of the form $(t_1, \ldots, t_n) \to t$, or $\perp$. The definition of *methodtype* is:

$$\frac{\begin{array}{l}\texttt{class}\ id\ \texttt{\{}\ \ldots\ m_1\ \ldots\ m_k\ \texttt{\}}\quad \text{is in the program}\\ \text{for some } j \in 1..k : methodname(m_j) = id^M\\ m_j \text{ is of the form}\\ \quad\texttt{public}\ t\ id^M\ \texttt{(}t_1^F\ id_1^F\texttt{,}\ \ldots\texttt{,}\ t_n^F\ id_n^F\texttt{)}\ \texttt{\{}t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_r\ id_r\texttt{;}\ s_1\ \ldots\ s_q\ \texttt{return}\ e\texttt{;}\ \texttt{\}}\end{array}}{methodtype(id, id^M) = (id_1^F : t_1^F, \ldots, id_n^F : t_n^F) \to t} \tag{16}$$

$$\frac{\begin{array}{l}\texttt{class}\ id\ \texttt{\{}\ \ldots\ m_1\ \ldots\ m_k\ \texttt{\}}\quad \text{is in the program}\\ \text{for all } j \in 1..k : methodname(m_j) \neq id^M\end{array}}{methodtype(id, id^M) = \perp} \tag{17}$$

$$\frac{\begin{array}{l}\texttt{class}\ id\ \texttt{extends}\ id^P\ \texttt{\{}\ \ldots\ m_1\ \ldots\ m_k\ \texttt{\}}\quad \text{is in the program}\\ \text{for some } j \in 1..k : methodname(m_j) = id^M\\ m_j \text{ is of the form}\\ \quad\texttt{public}\ t\ id^M\ \texttt{(}t_1^F\ id_1^F\texttt{,}\ \ldots\texttt{,}\ t_n^F\ id_n^F\texttt{)}\ \texttt{\{}t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_r\ id_r\texttt{;}\ s_1\ \ldots\ s_q\ \texttt{return}\ e\texttt{;}\ \texttt{\}}\end{array}}{methodtype(id, id^M) = (id_1^F : t_1^F, \ldots, id_n^F : t_n^F) \to t} \tag{18}$$

$$\frac{\begin{array}{l}\texttt{class}\ id\ \texttt{extends}\ id^P\ \texttt{\{}\ \ldots\ m_1\ \ldots\ m_k\ \texttt{\}}\quad \text{is in the program}\\ \text{for all } j \in 1..k : methodname(m_j) \neq id^M\end{array}}{methodtype(id, id^M) = methodtype(id^P, id^M)} \tag{19}$$

6.8 The *noOverloading* Helper function

$$\frac{methodtype(id^P, id^M) \neq \perp \quad \Rightarrow \quad methodtype(id^P, id^M) = methodtype(id, id^M)}{noOverloading(id, id^P, id^M)} \tag{20}$$

7 Type Rules

7.1 Type Judgments

We will use the following seven forms of type judgments:

$$\begin{array}{rcl} & \vdash & g \\ & \vdash & mc \\ & \vdash & d \\ C & \vdash & m \\ A, C & \vdash & s \\ A, C & \vdash & e : t \\ A, C & \vdash & p : t \end{array}$$

We can read a judgment as follows. The judgment $\vdash g$ means "the goal g type checks." The judgment $\vdash mc$ means "the main class mc type checks." The judgment $\vdash d$ means "the type declaration d type checks." The judgment $C \vdash m$ means "if defined in class C, the method declaration m type checks." The judgment $A, C \vdash s$ means "in a type environment A, if written in class C, the statement s type checks." The judgment $A, C \vdash e : t$ means "in a type environment A, if written in class C, the expression e has type t." The judgment $A, C \vdash p : t$ means "in a type environment A, if written in class C, the primary expression p has type t."

7.2 Goal

$$\frac{\begin{array}{l} \mathit{distinct}(\mathit{classname}(mc), \mathit{classname}(d_1), \ldots, \mathit{classname}(d_n)) \\ \mathit{acyclic}(\mathit{linkset}(d_1) \cup \ldots \cup \mathit{linkset}(d_n)) \\ \vdash mc \\ \vdash d_i \qquad i \in 1..n \end{array}}{\vdash mc\ d_1\ \ldots\ d_n} \tag{21}$$

7.3 Main Class

$$\frac{\mathit{distinct}(id_1, \ldots, id_r) \qquad [id_1 : t_1,\ \ldots, id_r : t_r], \bot \vdash s_i \qquad i \in 1..q}{\vdash \texttt{class}\ id\ \texttt{\{ public static void main (String [] }\ id^S\texttt{)}\{t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_r\ id_r\texttt{;}\ s_1 \ldots\ s_q\}\}} \tag{22}$$

7.4 Type Declarations

$$\frac{\begin{array}{l} \mathit{distinct}(id_1, \ldots, id_f) \\ \mathit{distinct}(\mathit{methodname}(m_1), \ldots, \mathit{methodname}(m_k)) \\ id \vdash m_i \qquad i \in 1..k \end{array}}{\vdash \texttt{class}\ id\ \texttt{\{}\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \texttt{\}}} \tag{23}$$

$$\frac{\begin{array}{l} \mathit{distinct}(id_1, \ldots, id_f) \\ \mathit{distinct}(\mathit{methodname}(m_1), \ldots, \mathit{methodname}(m_k)) \\ \mathit{noOverloading}(id, id^P, \mathit{methodname}(m_i)) \qquad id \vdash m_i \qquad i \in 1..k \end{array}}{\vdash \texttt{class}\ id\ \texttt{extends}\ id^P\ \texttt{\{}\ t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_f\ id_f\texttt{;}\ m_1\ \ldots\ m_k\ \texttt{\}}} \tag{24}$$

7.5 Method Declarations

$$\frac{\begin{array}{l} \mathit{distinct}(id_1^F, \ldots, id_n^F, id_1, \ldots, id_r) \\ A = \mathit{fields}(C) \cdot [id_1^F : t_1^F, \ldots, id_n^F : t_n^F, id_1 : t_1, \ldots, id_r : t_r] \\ A, C \vdash s_i \qquad i \in 1..q \qquad A, C \vdash e : t \end{array}}{C \vdash \texttt{public}\ t\ id^M\ \texttt{(}t_1^F\ id_1^F\texttt{,}\ \ldots\texttt{,}\ t_n^F\ id_n^F\texttt{)}\ \texttt{\{}t_1\ id_1\texttt{;}\ \ldots\texttt{;}\ t_r\ id_r\texttt{;}\ s_1\ \ldots\ s_q\ \texttt{return}\ e\texttt{;}\ \texttt{\}}} \tag{25}$$

7.6 Statements

$$\frac{A, C \vdash s_i \qquad i \in 1..q}{A, C \vdash \texttt{\{}\ s_1\ \ldots\ s_q\ \texttt{\}}} \tag{26}$$

$$\frac{A(id) = t_1 \qquad A, C \vdash e : t_2 \qquad t_2 \leq t_1}{A, C \vdash id\ \texttt{=}\ e\texttt{;}} \tag{27}$$

$$\frac{A(id) = \texttt{int[]} \qquad A, C \vdash e_1 : \texttt{int} \qquad A, C \vdash e_2 : \texttt{int}}{A, C \vdash id\ \texttt{[}\ e_1\ \texttt{]}\ \texttt{=}\ e_2\texttt{;}} \tag{28}$$

$$\frac{A, C \vdash e : \texttt{boolean} \qquad A, C \vdash s_1 \qquad A, C \vdash s_2}{A, C \vdash \texttt{if}\ \texttt{(}\ e\ \texttt{)}\ s_1\ \texttt{else}\ s_2} \tag{29}$$

$$\frac{A, C \vdash e : \texttt{boolean} \qquad A, C \vdash s}{A, C \vdash \texttt{while}\ \texttt{(}\ e\ \texttt{)}\ s} \tag{30}$$

$$\frac{A, C \vdash e : \texttt{int}}{A, C \vdash \texttt{System.out.println(}\ e\ \texttt{);}} \tag{31}$$

7.7 Expressions and Primary Expressions

$$\frac{A, C \vdash p_1 : \texttt{boolean} \qquad A, C \vdash p_2 : \texttt{boolean}}{A, C \vdash p_1 \ \texttt{\&\&} \ p_2 : \texttt{boolean}} \tag{32}$$

$$\frac{A, C \vdash p_1 : \texttt{int} \qquad A, C \vdash p_2 : \texttt{int}}{A, C \vdash p_1 \ \texttt{<} \ p_2 : \texttt{boolean}} \tag{33}$$

$$\frac{A, C \vdash p_1 : \texttt{int} \qquad A, C \vdash p_2 : \texttt{int}}{A, C \vdash p_1 \ \texttt{+} \ p_2 : \texttt{int}} \tag{34}$$

$$\frac{A, C \vdash p_1 : \texttt{int} \qquad A, C \vdash p_2 : \texttt{int}}{A, C \vdash p_1 \ \texttt{-} \ p_2 : \texttt{int}} \tag{35}$$

$$\frac{A, C \vdash p_1 : \texttt{int} \qquad A, C \vdash p_2 : \texttt{int}}{A, C \vdash p_1 \ \texttt{*} \ p_2 : \texttt{int}} \tag{36}$$

$$\frac{A, C \vdash p_1 : \texttt{int[]} \qquad A, C \vdash p_2 : \texttt{int}}{A, C \vdash p_1 \ \texttt{[} \ p_2 \ \texttt{]} : \texttt{int}} \tag{37}$$

$$\frac{A, C \vdash p : \texttt{int[]}}{A, C \vdash p \ \texttt{.length} : \texttt{int}} \tag{38}$$

$$\frac{\begin{array}{lll} A, C \vdash p : D & \mathit{methodtype}(D, id) = (t'_1, \ldots, t'_n) \to t & \\ A, C \vdash e_i : t_i & t_i \leq t'_i & i \in 1..n \end{array}}{A, C \vdash p \ \texttt{.}id \ \texttt{(}e_1\texttt{,} \ \ldots\texttt{,} \ e_n\texttt{)} : t} \tag{39}$$

$$A, C \vdash c : \texttt{int} \tag{40}$$

$$A, C \vdash \texttt{true} : \texttt{boolean} \tag{41}$$

$$A, C \vdash \texttt{false} : \texttt{boolean} \tag{42}$$

$$\frac{id \in \mathit{dom}(A)}{A, C \vdash id : A(id)} \tag{43}$$

$$\frac{C \neq \bot}{A, C \vdash \texttt{this} : C} \tag{44}$$

$$\frac{A, C \vdash e : \texttt{int}}{A, C \vdash \texttt{new int[}e\texttt{]} : \texttt{int[]}} \tag{45}$$

$$A, C \vdash \texttt{new} \ id\texttt{()} : id \tag{46}$$

$$\frac{A, C \vdash e : \texttt{boolean}}{A, C \vdash \texttt{!}e : \texttt{boolean}} \tag{47}$$

$$\frac{A, C \vdash e : t}{A, C \vdash \texttt{(}e\texttt{)} : t} \tag{48}$$