Languages-beta: MiniJava-Dynamics

The PLanCompS Project

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

Language "`MiniJava`"

1 Programs

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

MC : main-class ::= `class` identifier `{ public static void main ( String [ ]` identifier `) {` statemer

Semantics run⟦ P : program ⟧ : ⇒ null-type

Rule run⟦ `class` ID_1 `{ public static void main ( String [ ]` ID_2 `) {` S `} }` CD^* ⟧ =
scope(recursive(bound-names⟦ CD^* ⟧,
declare-classes⟦ CD^* ⟧),
execute⟦ S ⟧)

ID_1 and ID_2 are not referenced in S or CD^*

2 Declarations

Classes

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

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

Semantics bound-names⟦ CD^* : class-declaration* ⟧ : ⇒ sets(ids)
Rule bound-names⟦ `class` ID_1 `{` VD^* MD^* `}` ⟧ =
{id⟦ ID_1 ⟧}
Rule bound-names⟦ `class` ID_1 `extends` ID_2 `{` VD^* MD^* `}` ⟧ =
{id⟦ ID_1 ⟧}
Rule bound-names⟦ ⟧ =
{ }
Rule bound-names⟦ CD CD^+ ⟧ =
set-unite(bound-names⟦ CD ⟧,
bound-names⟦ CD^+ ⟧)

Semantics declare-classes⟦ CD^* : class-declaration* ⟧ : ⇒ envs
Rule declare-classes⟦ `class` ID_1 `{` VD^* MD^* `}` ⟧ =
{id⟦ ID_1 ⟧ ↦ class(thunk closure reference object(fresh-atom,
id⟦ ID_1 ⟧,
declare-variables⟦ VD^* ⟧),
declare-methods⟦ MD^* ⟧)}
Rule declare-classes⟦ `class` ID_1 `extends` ID_2 `{` VD^* MD^* `}` ⟧ =
{id⟦ ID_1 ⟧ ↦ class(thunk closure reference object(fresh-atom,
id⟦ ID_1 ⟧,
declare-variables⟦ VD^* ⟧,
dereference force class-instantiator bound id⟦ ID_2 ⟧),
declare-methods⟦ MD^* ⟧,
id⟦ ID_2 ⟧)}
Rule declare-classes⟦ ⟧ =
map( )
Rule declare-classes⟦ CD CD^+ ⟧ =
collateral(declare-classes⟦ CD ⟧,
declare-classes⟦ CD^+ ⟧)

Variables

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

Semantics declare-variables⟦ *VD** : var-declaration* ⟧ : ⇒ envs

Rule declare-variables⟦ *T ID* ; ⟧ =

{id⟦ *ID* ⟧ ↦ allocate-initialised-variable(type⟦ *T* ⟧,
initial-value⟦ *T* ⟧)}

Rule declare-variables⟦ ⟧ =

map( )

Rule declare-variables⟦ *VD VD*+ ⟧ =

collateral(declare-variables⟦ *VD* ⟧,
declare-variables⟦ *VD*+ ⟧)

Types

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

Semantics type⟦ *T* : type ⟧ : ⇒ types

Rule type⟦ `int [ ]` ⟧ =

vectors(variables)

Rule type⟦ `boolean` ⟧ =

booleans

Rule type⟦ `int` ⟧ =

integers

Rule type⟦ *ID* ⟧ =

pointers(objects)

Semantics initial-value⟦ *T* : type ⟧ : ⇒ minijava-values

Rule initial-value⟦ `int [ ]` ⟧ =

vector( )

Rule initial-value⟦ `boolean` ⟧ =

false

Rule initial-value⟦ `int` ⟧ =

0

Rule initial-value⟦ *ID* ⟧ =

pointer-null

Methods

Syntax *MD* : method-declaration ::= `public` type identifier `(` formal-list? `)` `{` var-declaration* statement* `r`

Type methods ⇝ functions(tuples(references(objects), minijava-values*), minijava-values)

Semantics declare-methods⟦ *MD** : method-declaration* ⟧ : ⇒ envs
Rule declare-methods⟦ `public` *T* *ID* `(` *FL*? `)` `{` *VD** *S** `return` *E* `;` `}` ⟧ =
{id⟦ *ID* ⟧ ↦ function closure scope(collateral(match(given,
tuple(pattern abstraction {"`this`" ↦ allocate-initialised-variable(pointers(objects)
given)},
bind-formals⟦ *FL*? ⟧)),
object-single-inheritance-feature-map checked dereference first tuple-elements give
declare-variables⟦ *VD** ⟧),
sequential(execute⟦ *S** ⟧,
evaluate⟦ *E* ⟧))}
Rule declare-methods⟦ ⟧ =
map( )
Rule declare-methods⟦ *MD* *MD*+ ⟧ =
collateral(declare-methods⟦ *MD* ⟧,
declare-methods⟦ *MD*+ ⟧)

Formals

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

Semantics bind-formals⟦ *FL*? : formal-list? ⟧ : ⇒ patterns*
Rule bind-formals⟦ *T* *ID* ⟧ =
pattern abstraction {id⟦ *ID* ⟧ ↦ allocate-initialised-variable(type⟦ *T* ⟧,
given)}
Rule bind-formals⟦ *T* *ID* `,` *FL* ⟧ =
bind-formals⟦ *T* *ID* ⟧,
bind-formals⟦ *FL* ⟧
Rule bind-formals⟦ ⟧ =
( )

3 Statements

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

Semantics execute⟦ S^* : statement* ⟧ : ⇒ null-type

Rule execute⟦ `{` S^* `}` ⟧ =
execute⟦ S^* ⟧

Rule execute⟦ `if (` E `)` S_1 `else` S_2 ⟧ =
if-true-else(evaluate⟦ E ⟧,
execute⟦ S_1 ⟧,
execute⟦ S_2 ⟧)

Rule execute⟦ `while (` E `)` S ⟧ =
while-true(evaluate⟦ E ⟧,
execute⟦ S ⟧)

Rule execute⟦ `System . out . println (` E `) ;` ⟧ =
print(to-string evaluate⟦ E ⟧,
"\n")

Rule execute⟦ ID `=` E `;` ⟧ =
assign(bound id⟦ ID ⟧,
evaluate⟦ E ⟧)

Rule execute⟦ ID `[` E_1 `] =` E_2 `;` ⟧ =
assign(checked index(integer-add(evaluate⟦ E_1 ⟧,
1),
vector-elements assigned bound id⟦ ID ⟧),
evaluate⟦ E_2 ⟧)

Rule execute⟦ ⟧ =
null

Rule execute⟦ S S^+ ⟧ =
sequential(execute⟦ S ⟧,
execute⟦ S^+ ⟧)

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 `)`

Type minijava-values ⇝ booleans | integers | vectors(variables) | pointers(objects)

Semantics evaluate⟦ *E* : expression ⟧ : ⇒ minijava-values

evaluate⟦ _ ⟧ is a well-typed funcon term only when _ is a well-typed MiniJava expression.

```
Rule evaluate⟦ E1 && E2 ⟧ =
        if-true-else(evaluate⟦ E1 ⟧,
          evaluate⟦ E2 ⟧,
          false)
Rule evaluate⟦ E1 < E2 ⟧ =
        integer-is-less(evaluate⟦ E1 ⟧,
          evaluate⟦ E2 ⟧)
Rule evaluate⟦ E1 + E2 ⟧ =
        integer-add(evaluate⟦ E1 ⟧,
          evaluate⟦ E2 ⟧)
Rule evaluate⟦ E1 - E2 ⟧ =
        integer-subtract(evaluate⟦ E1 ⟧,
          evaluate⟦ E2 ⟧)
Rule evaluate⟦ E1 * E2 ⟧ =
        integer-multiply(evaluate⟦ E1 ⟧,
          evaluate⟦ E2 ⟧)
Rule evaluate⟦ E1 [ E2 ] ⟧ =
        assigned checked index(integer-add(evaluate⟦ E2 ⟧,
                  1),
                vector-elements evaluate⟦ E1 ⟧)
Rule evaluate⟦ E . length ⟧ =
        length vector-elements evaluate⟦ E ⟧
Rule evaluate⟦ E . ID ( EL? ) ⟧ =
        give(evaluate⟦ E ⟧,
          apply(lookup(class-name-single-inheritance-feature-map object-class-name checked dereference
              id⟦ ID ⟧),
            tuple(given,
              evaluate-actuals⟦ EL? ⟧)))
Rule evaluate⟦ IL ⟧ =
       integer-value⟦ IL ⟧
Rule evaluate⟦ true ⟧ =
        true
Rule evaluate⟦ false ⟧ =
        false
Rule evaluate⟦ ID ⟧ =
        assigned bound id⟦ ID ⟧
Rule evaluate⟦ this ⟧ =
        assigned bound "this" 7
Rule evaluate⟦ new int [ E ] ⟧ =
        vector(interleave-repeat(allocate-initialised-variable(integers,
              0),
            1,
            evaluate⟦ E ⟧))
Rule evaluate⟦ new ID ( ) ⟧ =
```

Syntax *EL* : expression-list ::= expression (, expression-list)?

Semantics evaluate-actuals⟦ *EL*? : expression-list? ⟧ : (⇒ minijava-values)*
Rule evaluate-actuals⟦ *E* ⟧ =
evaluate⟦ *E* ⟧
Rule evaluate-actuals⟦ *E* , *EL* ⟧ =
evaluate⟦ *E* ⟧,
evaluate-actuals⟦ *EL* ⟧
Rule evaluate-actuals⟦ ⟧ =
( )

5 Lexemes

Lexis *ID* : identifier ::= letter (letter | digit | _)*

Semantics id⟦ *ID* : identifier ⟧ : ⇒ ids = "*ID*"

Lexis *IL* : integer-literal ::= digit+
letter ::= a – z
| A – Z
digit ::= 0 – 9

Semantics integer-value⟦ *IL* : integer-literal ⟧ : ⇒ integers = decimal-natural "*IL*"