More Formal Semantics of Java Types

Range of primitive integer types in Java

Type	Range	Bits
`byte`	$[-128, 127]$	**8**
`short`	$[-32768, 32767]$	**16**
`int`	$[-2147483648, 2147483647]$	**32**
`long`	$[-2^{63}, 2^{63} - 1]$	**64**

Examples

Valid for Java integer semantics

$\texttt{MAX\_INT+1} \doteq \texttt{MIN\_INT}$

$\texttt{MIN\_INT*}(-1) \doteq \texttt{MIN\_INT}$

$\exists \texttt{x,y:int.x} \not\doteq 0 \wedge \texttt{y} \not\doteq 0 \wedge \texttt{x*y} \doteq 0$

Examples

Valid for Java integer semantics

$\texttt{MAX\_INT+1} \doteq \texttt{MIN\_INT}$

$\texttt{MIN\_INT*}(-1) \doteq \texttt{MIN\_INT}$

$\exists \texttt{x,y:int.x} \not\doteq 0 \wedge \texttt{y} \not\doteq 0 \wedge \texttt{x*y} \doteq 0$

Not valid for Java integer semantics

$\forall \texttt{x:int.} \exists \texttt{y:int.y} > \texttt{x}$

Examples

Valid for Java integer semantics

$$\texttt{MAX\_INT+1} \doteq \texttt{MIN\_INT}$$

$$\texttt{MIN\_INT*}(-1) \doteq \texttt{MIN\_INT}$$

$$\exists \texttt{x,y:int.x} \not\doteq 0 \wedge \texttt{y} \not\doteq 0 \wedge \texttt{x*y} \doteq 0$$

Not valid for Java integer semantics

$$\forall \texttt{x:int.} \exists \texttt{y:int.y} > \texttt{x}$$

Not a sound rewrite rules for Java integer semantics

$$\texttt{x+1} > \texttt{y+1} \quad \leadsto \quad \texttt{x} > \texttt{y}$$

General Problem revisited

- semantic gap between $\mathbb{Z}$ and Java integers
- defining a JavaDL semantics for Java integers that...
  - is a correct data refinement of $\mathbb{Z}$ **Req. 1**
  - reflects Java integer semantics **Req. 2**

General Problem revisited

- semantic gap between $\mathbb{Z}$ and Java integers
- defining a JavaDL semantics for Java integers that...
  - is a correct data refinement of $\mathbb{Z}$ **Req. 1**
  - reflects Java integer semantics **Req. 2**

3 approaches

Semantics	Description	Req. 1	Req. 2
$\mathcal{S}_{OCL}$	corresponds to semantics of $\mathbb{Z}$	√	X
$\mathcal{S}_{Java}$	corresponds to Java semantics	X	√
$\mathcal{S}_{KeY}$	combination of $\mathcal{S}_{OCL}$ and $\mathcal{S}_{Java}$	√	√

Semantics $\mathcal{S}_{OCL}$

$\mathcal{S}_{OCL}$ **assigns Java integers semantics of** $\mathbb{Z}$

- **Req. 1 trivially fulfilled**
- **Req. 2 violated, incorrect programs can be “verified”**

Example:

$$\models_{\mathcal{S}_{OCL}} \forall \texttt{x:int.}\langle \texttt{y=x+1;}\rangle \texttt{y} \doteq \texttt{x} +_{\mathbb{Z}} 1$$

but for $\texttt{x} \doteq \texttt{MAX\_INT}$ **program not correct**

Semantics $\mathcal{S}_{Java}$

$\mathcal{S}_{Java}$ **assigns Java integers the semantics defined in Java Language Specification**

- **Req. 1 violated**

  several abstract states mapped onto one concrete state
- **Req. 2 trivially fulfilled**

No incorrect programs can be verified, but

violation of Req. 1 leads to “incidentally” correct programs!

Our Approach

Approach

- **types** `byte`, `short`, `int`, `long` **have semantics** $\mathcal{S}_{Java}$
- **additional virtual types** `arithByte`, `arithShort`, `arithInt`, **and** `arithLong` **(called “arithmetical types”) with following semantics:**

Our Approach

Approach

- **types** `byte`, `short`, `int`, `long` **have semantics** $\mathcal{S}_{Java}$
- **additional virtual types** `arithByte`, `arithShort`, `arithInt`, **and** `arithLong` **(called "arithmetical types") with following semantics:**
  - **arithmetical types have infinite range**
  - **operators are underspecified:**

    Semantics as in $\mathbb{Z}$ but semantics unspecified if

    both arguments are in valid range but result is not.

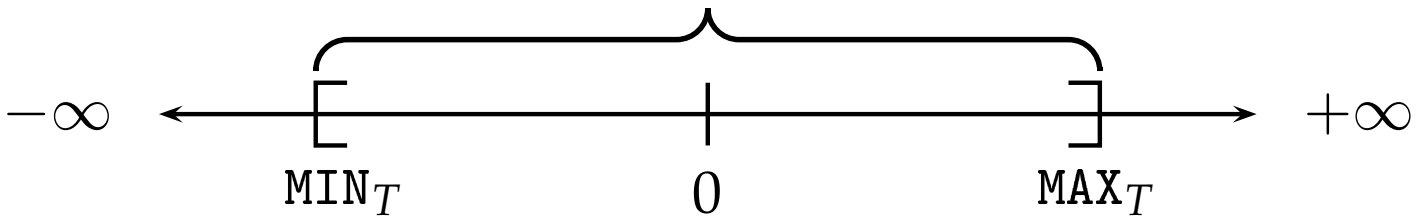
Our Approach

Approach

- **types** `byte`, `short`, `int`, `long` **have semantics** $\mathcal{S}_{Java}$
- **additional virtual types** `arithByte`, `arithShort`, `arithInt`, **and** `arithLong` **(called "arithmetical types") with following semantics:**
  - **arithmetical types have infinite range**
  - **operators are underspecified:**

    Semantics as in $\mathbb{Z}$ but semantics unspecified if

    both arguments are in valid range but result is not.



Semantics $\mathcal{S}_{KeY}$

Semantics $\mathcal{S}_{KeY}$ of `arithByte`, `arithShort`, `arithInt`, `arithLong`

Range: infinite ($\mathbb{Z}$)

Operations, e.g. `+` on `arithInt`:

$\texttt{+} : \texttt{arithInt} \times \texttt{arithInt} \longrightarrow \texttt{arithInt}$

3 cases:

Semantics $\mathcal{S}_{KeY}$

Semantics $\mathcal{S}_{KeY}$ of `arithByte`, `arithShort`, `arithInt`, `arithLong`

Range: infinite ($\mathbb{Z}$)

Operations, e.g. `+` on `arithInt`:

$\texttt{+} : \texttt{arithInt} \times \texttt{arithInt} \longrightarrow \texttt{arithInt}$

3 cases:

- **both args. and result are in valid range (e.g. $2\texttt{+}3 \doteq 5$)**

  "normal" case

Semantics $\mathcal{S}_{KeY}$

Semantics $\mathcal{S}_{KeY}$ of `arithByte`, `arithShort`, `arithInt`, `arithLong`

Range: infinite ($\mathbb{Z}$)

Operations, e.g. `+` on `arithInt`:

$\texttt{+} : \texttt{arithInt} \times \texttt{arithInt} \longrightarrow \texttt{arithInt}$

3 cases:

- **both args. and result are in valid range (e.g. $2\texttt{+}3 \doteq 5$)**

  "normal" case

- **both args. are in valid range but result is not (e.g. $\texttt{MAX\_INT+}1 \doteq$?)**

  overflow case, not specified

Semantics $\mathcal{S}_{KeY}$

Semantics $\mathcal{S}_{KeY}$ of `arithByte`, `arithShort`, `arithInt`, `arithLong`

Range: infinite ($\mathbb{Z}$)

Operations, e.g. `+` on `arithInt`:

$\texttt{+} : \texttt{arithInt} \times \texttt{arithInt} \longrightarrow \texttt{arithInt}$

3 cases:

- **both args. and result are in valid range (e.g. $2\texttt{+}3 \doteq 5$)**

  "normal" case

- **both args. are in valid range but result is not (e.g. $\texttt{MAX\_INT}\texttt{+}1 \doteq$?)**

  overflow case, not specified

- **an arg. is not in valid range (e.g. $(\texttt{MAX\_INT} +_{\mathbb{Z}} 1)\texttt{+}1 \doteq \texttt{MAX\_INT} +_{\mathbb{Z}} 2$)**

  cannot happen during execution, only in logic

Reason for Underspecification

property provable in our calculus $\Longrightarrow$

property independent of actual implementation of overflow case

Reason for Underspecification

property provable in our calculus $\Longrightarrow$

property independent of actual implementation of overflow case

Main theorem

If

(i) $\models_{\mathcal{S}_{KeY}} \Gamma \rightarrow \langle p \rangle \psi$

(ii) $s \models_{\mathcal{S}_{KeY}} \Gamma$

(iii) s **is a real state (i.e. all arith. variables in valid range)**

then

(a) no overflow occurs in p' **when started in** s'

(b) p' **terminates in state where property** ψ **holds.**

A Sequent Calculus

Generation of pre-conditions that no overflow occurs built into calculus rules

Example: Rule for multiplication on arithmetical types

define predicate $in_{\mathtt{T}}(\cdot)$**:** $in_{\mathtt{T}}(\mathtt{x})$ ***iff*** $\mathtt{MIN\_}T \leq \mathtt{x} \leq \mathtt{MAX\_}T$

$$\frac{1.\ \Gamma,\ in_{\mathtt{T}_1}(\mathtt{x}) \wedge in_{\mathtt{T}_2}(\mathtt{y}) \rightarrow in_{\mathtt{T}}(\mathtt{x} * \mathtt{y}) \vdash \{\mathtt{z} \leftarrow \mathtt{x} * \mathtt{y}\}\langle\rangle\phi \qquad 2.\ \Gamma,\ in_{\mathtt{T}_1}(\mathtt{x}),\ in_{\mathtt{T}_2}(\mathtt{y}),\ \neg in_{\mathtt{T}}(\mathtt{x} * \mathtt{y}) \vdash \langle \mathtt{z = overflow(x, y, "*");}\ \rangle\phi}{\Gamma \vdash \langle \mathtt{z = x*y;}\ \rangle\phi}$$

Software Development

Software development following our approach:

- *Specification*: use of OCL type INTEGER
- *Implementation*: use of arithmetical types (e. g. `arithInt`)
- *Verification*: if all proof obligations are provable in our calculus
  - specified properties hold
  - no overflow occurs
- arithmetical types can safely be replaced by corresponding Java types