Formal Verification of Software

Dynamic Logic for Java

Bernhard Beckert

Universität Koblenz-Landau
KeY Supports Java Card as Target Language

What is Java Card?

- Subset of Java
- Sun’s official standard for SMARTCARDS and embedded devices
KeY Supports Java Card as Target Language

What is Java Card?

- Subset of Java
- Sun’s official standard for SMARTCARDS and embedded devices

Why Java Card?
KeY Supports Java Card as Target Language

What is Java Card?

- Subset of Java
- Sun’s official standard for SMARTCARDS and embedded devices

Why Java Card?
Good example for real-world object-oriented language
KeY Supports Java Card as Target Language

What is Java Card?

- Subset of Java
- Sun’s official standard for SMARTCARDS and embedded devices

Why Java Card?
Good example for real-world object-oriented language

Java Card has no
- garbage collection
- dynamical class loading
- multi-threading
- floating-point arithmetic
KeY Supports Java Card as Target Language

What is Java Card?

- Subset of Java
- Sun’s official standard for SMARTCARDS and embedded devices

Why Java Card?

Good example for real-world object-oriented language

Java Card has no
- garbage collection
- dynamical class loading
- multi-threading
- floating-point arithmetic

Application Area
- security critical
- financial risk
  (e.g. exchanging smart cards is expensive)
Academic vs. Real-world Languages

Problems to address

Pointers / objects attributes

Modelled as non-rigid constants and functions
Academic vs. Real-world Languages

Problems to address

Pointers / objects attributes

  Modelled as non-rigid constants and functions

Side effects

  Expressions in programs have side effects, for example

    if ((y=3) + y < 0) .. else ..
Academic vs. Real-world Languages

Problems to address

Pointers / objects attributes

Modelled as non-rigid constants and functions

Side effects

Expressions in programs have side effects, for example

\[
\text{if } ((y=3) + y < 0) \quad \text{else } \quad .
\]

Aliasing

Different names may refer to the same location, for example

\[
o.a, u.a \quad \text{in a state } g \quad \text{where } g \models o \triangleq u
\]
Other Issues (Later)

Further supported Java Card features

- method invocation, dynamic binding
Other Issues (Later)

Further supported Java Card features

- method invocation, dynamic binding
- polymorphism
Other Issues (Later)

Further supported Java Card features

- method invocation, dynamic binding
- polymorphism
- abrupt termination
Other Issues (Later)

Further supported Java Card features

- method invocation, dynamic binding
- polymorphism
- abrupt termination
- checking for nullpointer exceptions
Other Issues (Later)

Further supported Java Card features

- method invocation, dynamic binding
- polymorphism
- abrupt termination
- checking for nullpointer exceptions
- object creation and initialisation
Other Issues (Later)

Further supported Java Card features

- method invocation, dynamic binding
- polymorphism
- abrupt termination
- checking for nullpointer exceptions
- object creation and initialisation
- arrays
Other Issues (Later)

Further supported Java Card features

- method invocation, dynamic binding
- polymorphism
- abrupt termination
- checking for nullpointer exceptions
- object creation and initialisation
- arrays
- finiteness of integer data types
Other Issues (Later)

Further supported Java Card features

- method invocation, dynamic binding
- polymorphism
- abrupt termination
- checking for nullpointer exceptions
- object creation and initialisation
- arrays
- finiteness of integer data types
- transactions
Handling Object Attributes

Similar concepts

- Object attributes
- Arrays
- Pointers
Handling Object Attributes

Similar concepts

- Object attributes
- Arrays
- Pointers

Non-rigid functions

Attributes are considered to be non-rigid functions on objects
Handling Object Attributes

Similar concepts

- Object attributes
- Arrays
- Pointers

Non-rigid functions

Attributes are considered to be non-rigid functions on objects

Extended to program variables

Program variables are considered to be non-rigid constants
Side Effects: Symbolic Execution Paradigm

Expressions may have side effects, for example a simple assignment

\[(y=3) + y < 0\]

does not only evaluate to a boolean value, but also assigns a value to \(y\).
Expressions may have side effects, for example a simple assignment

\[(y=3) + y < 0\]

does not only evaluate to a boolean value, but also assigns a value to \(y\).

**Problem:** Terms in logic have to be side effect free
Side Effects: Symbolic Execution Paradigm

Expressions may have side effects, for example a simple assignment

\[(y=3) + y < 0\]

does not only evaluate to a boolean value, but also assigns a value to \(y\).

**Problem:** Terms in logic have to be side effect free

**Solution:**

- Calculus rules realise a stepwise symbolic execution of the programs (program transformation)
Side Effects: Symbolic Execution Paradigm

Expressions may have side effects, for example a simple assignment

\[(y=3) + y < 0\]

does not only evaluate to a boolean value, but also assigns a value to \(y\).

**Problem:** Terms in logic have to be side effect free

**Solution:**

- Calculus rules realise a stepwise symbolic execution of the programs (program transformation)

- Restrict applicability of some rules. For example, \texttt{if-then-else} is only applicable, if the guard is free of side-effects
Rule Application for if-then-else

\[ \Gamma \vdash \langle \text{if } ((y = 3) + y < 0)\{\alpha}\ \text{else}\{\beta}\rangle \Phi, \Delta \]
Rule Application for \textit{if-then-else}

\[ \Gamma \vdash \langle \text{boolean guard} = (y = 3) + y < 0; \text{if (guard)}\{\alpha}\ \text{else}\{\beta}\rangle \Phi, \Delta \]

\[ \Gamma \vdash \langle \text{if } ((y = 3) + y < 0)\{\alpha}\ \text{else}\{\beta}\rangle \Phi, \Delta \]
Rule Application for \texttt{if-then-else}

\[\Gamma \vdash \begin{cases} \text{int val0} = (y = 3) + y; \\ \text{boolean guard} = \text{val0} < 0; \\ \text{if (guard)}\{\alpha}\text{ else}\{\beta}\end{cases} \Phi, \Delta\]

\[\Gamma \vdash \langle \text{boolean guard} = (y = 3) + y < 0; \text{if (guard)}\{\alpha}\text{ else}\{\beta}\rangle \Phi, \Delta\]

\[\Gamma \vdash \langle \text{if ((y = 3) + y < 0)}\{\alpha}\text{ else}\{\beta}\rangle \Phi, \Delta\]
Rule Application for \textit{if-then-else}

\begin{align*}
\Gamma & \vdash \left\langle \begin{array}{l}
\text{int } \text{val1 } = y = 3; \\
\text{int } \text{val0 } = \text{val1 } + y
\end{array} \right\rangle \Phi, \Delta \\
\Gamma & \vdash \left\langle \begin{array}{l}
\text{int } \text{val0 } = (y = 3) + y; \\
\text{boolean } \text{guard } = \text{val0 } < 0;
\end{array} \right\rangle \Phi, \Delta \\
&& \text{if (guard)} \{\alpha\} \text{ else } \{\beta\} \\
\Gamma & \vdash \left\langle \text{boolean } \text{guard } = (y = 3) + y < 0; \text{ if (guard)} \{\alpha\} \text{ else } \{\beta\} \right\rangle \Phi, \Delta \\
\Gamma & \vdash \left\langle \text{if } ((y = 3) + y < 0) \{\alpha\} \text{ else } \{\beta\} \right\rangle \Phi, \Delta
\end{align*}
Rule Application for if-then-else

\[ \begin{align*}
\Gamma & \vdash \begin{cases}
y = 3; \\
\text{int val1} = y; \\
\text{int val0} = \text{val1} + y
\end{cases} \Phi, \Delta \\
... \\
\end{align*} \]

\[ \begin{align*}
\Gamma & \vdash \begin{cases}
y = 3; \\
\text{int val1} = y + 3; \\
\text{int val0} = \text{val1} + y
\end{cases} \Phi, \Delta \\
... \\
\end{align*} \]

\[ \begin{align*}
\Gamma & \vdash \begin{cases}
\text{int val0} = (y = 3) + y; \\
\text{boolean guard} = \text{val0} < 0; \\
\text{if (guard)} \{ \alpha \} \text{ else } \{ \beta \}
\end{cases} \Phi, \Delta \\
\end{align*} \]

\[ \begin{align*}
\Gamma & \vdash \langle \text{boolean guard} = (y = 3) + y < 0; \text{ if (guard)} \{ \alpha \} \text{ else } \{ \beta \} \rangle \Phi, \Delta \\
\end{align*} \]
Assignment in the Classical Version

Classical rule for assignment

\[
\begin{align*}
\Gamma^{x\leftarrow y}, x \doteq t^{x\leftarrow y} \vdash \Phi, \Delta^{x\leftarrow y} \\
\Gamma \vdash \langle x = t \rangle \Phi, \Delta
\end{align*}
\]
(y new variable)
Assignment in the Classical Version

Classical rule for assignment

\[
\Gamma^{x \leftarrow y}, x \equiv t^{x \leftarrow y} \vdash \Phi, \Delta^{x \leftarrow y} \quad (y \text{ new variable})
\]

\[
\Gamma \vdash \langle x = t \rangle \Phi, \Delta
\]

Problems:

- cannot be handled as substitution
Assignment in the Classical Version

Classical rule for assignment

\[
\frac{\Gamma^{\tilde{x} \leftarrow y}, x \equiv t^{\tilde{x} \leftarrow y} \vdash \Phi, \Delta^{\tilde{x} \leftarrow y}}{\Gamma \vdash \langle x = t \rangle \Phi, \Delta} (y \text{ new variable})
\]

Problems:

- *cannot* be handled as substitution

- aliasing:
  
  \[
  \frac{?}{o.a \equiv 3 \vdash \langle u.a = 5; \rangle \phi}
  \]

Requires to split the proof for the cases \( o = u \) and \( o \neq u \).
The Active Statement in a Program

Example

\[
1:\{\text{try}\{\ i=0; \ \ j=0; \ \}\ \text{finally}\{\ k=0; \ \}\}\}
\]

first active command \( i=0; \)
non-active prefix \( \pi \)
rest \( \omega \)
Updates: Delayed Substitutions

**Syntax**: Updates are syntactical elements

\[ \{loc := val\} \Phi \lor \{loc := val\} t \]

where

- **loc** either a
  - program variable \(x\)
  - an attribute \(o.attr\) or
  - an array access \(a[i]\)

- **val** a logical term (no side effects)
Updates: Delayed Substitutions

Syntax: Updates are syntactical elements

{loc := val}Φ or {loc := val}t

where

loc either a
- program variable x
- an attribute o.attr or
- an array access a[i]

val a logical term (no side effects)

Semantic:

\[ g \models \{loc := val\}Φ \quad \text{iff} \quad g' \models Φ \text{ where } g' = g_{loc}^{val} \]
Assignment Rule in KeY

\[
\Gamma \vdash \{loc := val\}(\pi \omega)\Phi, \Delta \quad \text{where \(loc, val\) side effect free}
\]

\[
\Gamma \vdash (\pi loc = val; \omega)\Phi, \Delta
\]
Assignment Rule in KeY

\[
\Gamma \vdash \{loc := val\}(\pi \omega)\Phi, \Delta \\
\hline
\Gamma \vdash (\pi loc = val; \omega)\Phi, \Delta
\]

where \(loc, val\) side effect free

Advantages:

- no renaming as in the classical version
Assignment Rule in KeY

\[ \Gamma \vdash \{ \text{loc := val} \} \langle \pi \omega \rangle \Phi, \Delta \]

, where loc, val side effect free

\[ \Gamma \vdash \langle \pi \text{loc = val}; \ \omega \rangle \Phi, \Delta \]

Advantages:

- no renaming as in the classical version

- delayed proof branching

\[ \Gamma \vdash \langle x = 3; \ x = 4; \rangle \Phi \quad \text{or} \]

\[ \Gamma \vdash \langle o.a = 3; \ o.a = 4; \rangle \Phi \]
Use conditional terms to delay splitting further

\[(s[t_1 \ ? = t_2] \mapsto e)^{I,\beta} = \begin{cases} e^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ (s[t_1])^{I,\beta} & \text{otherwise} \end{cases}\]
Application of updates $\mathcal{U}$

Application on

program variable

\[
\begin{align*}
\{x := t\} y & \rightsquigarrow y \\
\{x := t\} x & \rightsquigarrow t \\
\{o.a := t\} y & \rightsquigarrow y
\end{align*}
\]
## Application of updates $\mathcal{U}$

<table>
<thead>
<tr>
<th>Application on program variable</th>
<th>Application on attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x := t} \ y \leadsto y$</td>
<td>${o.a := t} \ o.a \leadsto t$</td>
</tr>
<tr>
<td>${x := t} \ x \leadsto t$</td>
<td>${o.a := t} \ u.a \leadsto ({o.a := t} u? = o).a \mapsto t$</td>
</tr>
<tr>
<td>${o.a := t} \ y \leadsto y$</td>
<td></td>
</tr>
</tbody>
</table>
## Application of updates \( \mathcal{U} \)

<table>
<thead>
<tr>
<th>Application on program variable</th>
<th>Application on attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ x := t } y \xrightarrow{} y</td>
<td>{ o.a := t } o.a \xrightarrow{} t</td>
</tr>
<tr>
<td>{ x := t } x \xrightarrow{} t</td>
<td>{ o.a := t } u.a \xrightarrow{} ({ o.a := t } u? = o).a \leftrightarrow t</td>
</tr>
<tr>
<td>{ o.a := t } y \xrightarrow{} y</td>
<td></td>
</tr>
</tbody>
</table>

Application stops before modal operators, e.g.

\{ o.a := t \} \langle \alpha \rangle \Phi \xrightarrow{} \{ o.a := t \} \langle \alpha \rangle \Phi

Application is shoved over operators to the subformulas (terms)

\{ o.a := t \} \Phi \land \Psi \xrightarrow{} \{ o.a := t \} \Phi \land \{ o.a := t \} \Psi
## Application of updates \( U \)

<table>
<thead>
<tr>
<th>Application on program variable</th>
<th>Application on attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {x := t} \ y \leadsto y )</td>
<td>( {o.a := t} \ o.a \leadsto t )</td>
</tr>
<tr>
<td>( {x := t} \ x \leadsto t )</td>
<td>( {o.a := t} \ u.a \leadsto ({o.a := t}u? = o).a \leftrightarrow t )</td>
</tr>
<tr>
<td>( {o.a := t} \ y \leadsto y )</td>
<td></td>
</tr>
</tbody>
</table>

### Example

\( \{o.a := o\} o.a.a.b \)
## Application of updates $\mathcal{U}$

<table>
<thead>
<tr>
<th>Application on program variable</th>
<th>Application on attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x := t} y \leadsto y$</td>
<td>${o.a := t} o.a \leadsto t$</td>
</tr>
<tr>
<td>${x := t} x \leadsto t$</td>
<td>${o.a := t} u.a \leadsto ({o.a := t} u? = o).a \mapsto t$</td>
</tr>
<tr>
<td>${o.a := t} y \leadsto y$</td>
<td></td>
</tr>
</tbody>
</table>

### Example

$\{o.a := o\} o.a.a.b \leadsto \{o.a := o\} o.a.a.b$
Application of updates \( \mathcal{U} \)

<table>
<thead>
<tr>
<th>Application on program variable</th>
<th>Application on attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>( { x := t } \ y \rightsquigarrow y )</td>
<td>( { o.a := t } \ o.a \rightsquigarrow t )</td>
</tr>
<tr>
<td>( { x := t } \ x \rightsquigarrow t )</td>
<td>( { o.a := t } \ u.a \rightsquigarrow ({ o.a := t } u? = o).a \mapsto t )</td>
</tr>
<tr>
<td>( { o.a := t } \ y \rightsquigarrow y )</td>
<td></td>
</tr>
</tbody>
</table>

Example

\( \{ o.a := o \} o.a.a\cdot b \rightsquigarrow (\{ o.a := o \} o.a.a).b \)
Application of updates $\mathcal{U}$

<table>
<thead>
<tr>
<th>Application on program variable</th>
<th>Application on attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x := t} y \leadsto y$</td>
<td>${o.a := t} o.a \leadsto t$</td>
</tr>
<tr>
<td>${x := t} x \leadsto t$</td>
<td>${o.a := t} u.a \leadsto ({o.a := t}u? = o).a \mapsto t$</td>
</tr>
<tr>
<td>${o.a := t} y \leadsto y$</td>
<td></td>
</tr>
</tbody>
</table>

Example

$\{o.a := o\} o.a.a.b \leadsto (\{o.a := o\} o.a? = o).a \mapsto o).b$
Application of updates \( U \)

<table>
<thead>
<tr>
<th>Application on program variable</th>
<th>Application on attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x := t} y \leadsto y</td>
<td>{o.a := t} o.a \leadsto t</td>
</tr>
<tr>
<td>{x := t} x \leadsto t</td>
<td>{o.a := t} u.a \leadsto ({o.a := t}u? = o).a \leftarrow t</td>
</tr>
<tr>
<td>{o.a := t} y \leadsto y</td>
<td></td>
</tr>
</tbody>
</table>

Example

\{o.a := o\}o.a.a.b \leadsto (o? = o).a \leftarrow o).b
Application of updates $U$

Application on program variable

- $\{x := t\} y \rightsquigarrow y$
- $\{x := t\} x \rightsquigarrow t$
- $\{o.a := t\} y \rightsquigarrow y$

Application on attribute

- $\{o.a := t\} o.a \rightsquigarrow t$
- $\{o.a := t\} u.a \rightsquigarrow (\{o.a := t\}u?. = o).a \leftrightarrow t$

Example

- $\{o.a := o\} o.a.a.b \rightsquigarrow (o?. = o).a \leftrightarrow o).b$
Application of updates $\mathcal{U}$

<table>
<thead>
<tr>
<th>Application on program variable</th>
<th>Application on attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x := t} y \leadsto y$</td>
<td>${o.a := t} o.a \leadsto t$</td>
</tr>
<tr>
<td>${x := t} x \leadsto t$</td>
<td>${o.a := t} u.a \leadsto ({o.a := t}u? = o).a \leftarrow t$</td>
</tr>
<tr>
<td>${o.a := t} y \leadsto y$</td>
<td></td>
</tr>
</tbody>
</table>

Example

$\{o.a := o\} o.a.a.b \leadsto o.b$
Parallel Updates

Computing update followed by update

\[
\{l_1 := r_1\}\{l_2 := r_2\} = \{\{l_1 := r_1\}, \{l_1 := r_1\} \downarrow l_2 := \{l_1 := r_1\}r_2\}\]

where \(u \downarrow l = \begin{cases} x & \text{if } l = x \text{ is a program variable} \\ (\overline{u} u).a & \text{if } l = u.a \end{cases}\)

Results in parallel update: \(\{l_1 := v_1, \ldots, l_n := v_n\}\)
Parallel Updates

Computing update followed by update

\[
\{l_1 := r_1\}\{l_2 := r_2\} = \{\{l_1 := r_1\}, \{l_1 := r_1\} \downarrow l_2 := \{l_1 := r_1\}r_2\}
\]

where

\[
u \downarrow l = \begin{cases} x & \text{if } l = x \text{ is a program variable} \\ (\forall u).a & \text{if } l = u.a \end{cases}
\]

Results in parallel update: \(\{l_1 := v_1, \ldots, l_n := v_n\}\)

Semantics

- All \(l_i\) and \(v_i\) computed in old state
- All updates done simultaneously
- If conflict \(l_i = l_j, v_i \neq v_j\) later update wins
Quantifying over Program Variables

Cannot quantify over program variables (non-rigid constants)

Non allowed: $\forall i:\text{int} (\langle \alpha(i) \rangle F)$

Non allowed: $\forall n (\langle \alpha(n) \rangle F)$
Quantifying over Program Variables

Cannot quantify over program variables (non-rigid constants)

Non allowed: \( \forall i : \text{int} (\langle \alpha(i) \rangle F) \)

Non allowed: \( \forall n (\langle \alpha(n) \rangle F) \)

Solution

\( \forall n \{ i := n \} (\langle \alpha(i) \rangle F) \)
Abrupt Changes of the Control Flow

Abrupt Termination: Redirection of the control flow by

return, break, continue or Exceptions
Abrupt Changes of the Control Flow

**Abrupt Termination:** Redirection of the control flow by

```
return, break, continue or Exceptions
```

```
<try{
    a = a/b;
    a = a + 1;
}
catch(Exception e){...}
finally {...} Φ
```

**Decomposition Rule**

not applicable
Abrupt Changes of the Control Flow

**Abrupt Termination:** Redirection of the control flow by

\[
\text{return, break, continue or Exceptions}
\]

\[
\langle \text{try} \{
    a = a / b;
    a = a + 1;
\} \text{catch(Exception e)} \{...\}
\]

**Solution:** The rules work on the first active statement

\[
\Gamma \vdash \langle \pi \text{ stmt}'; \ \omega \rangle \Phi, \Delta
\]

\[
\Gamma \vdash \langle \pi \text{ stmt}; \ \omega \rangle \Phi, \Delta
\]
Catch Thrown Exception

Rule

\[ \Gamma \vdash \langle \text{try}\{ \text{throw exc; p} \} \quad \text{catch (Exception e)} \{ q \} \quad \text{finally}\{ r \} \rangle \Phi, \Delta \]
Catch Thrown Exception

Rule

\[
\Gamma \vdash \langle \text{if (exc instanceof Exception)} \rangle \left\{ \begin{array}{l}
\text{try}\{e = \text{exc}; q\} \text{finally}\{r\}\\
\text{else}\{r \text{ throw exc}; \}\end{array}\right\} \Phi, \Delta
\]

\[
\Gamma \vdash \langle \text{try}\{\text{throw exc; p}\} \rangle \left\langle \begin{array}{l}
\text{catch (Exception e)}\{q\}\\
\text{finally}\{r\}\end{array}\right\} \Phi, \Delta
\]