Introduction to JavaCard Dynamic Logic

Andreas Roth, Richard Bubel, Christian Engel

December 12, 2007

Some Java Features

- Assignments, **complex** expressions with **side effects**
  `int i = 0; if ((i=2) >= 2) {i++;}` // value of i?

Some Java Features

- Assignments, **complex** expressions with **side effects**
  `int i = 0; if ((i=2) >= 2) {i++;}` // value of i?
- Abrupt termination

Some Java Features

- Assignments, **complex** expressions with **side effects**
  `int i = 0; if ((i=2) >= 2) {i++;} //` value of i?
- Abrupt termination
  - **Exceptions** (`try` – `catch` – `finally`)

Some Java Features

- Assignments, **complex** expressions with **side effects**
  `int i = 0; if ((i=2) >= 2) {i++;} //` value of i?
- Abrupt termination
  - **Exceptions** (`try` – `catch` – `finally`)
  - **Local jumps** `return`, `break`, `continue`

Some Java Features

- Assignments, **complex** expressions with **side effects**
  `int i = 0; if ((i=2) >= 2) {i++;}` `//` value of i?
- Abrupt termination
  - **Exceptions** (`try` – `catch` – `finally`)
  - **Local jumps** `return`, `break`, `continue`
- **Aliasing**
  Different navigation expressions may be same object reference

$$I \models \mathrm{o.age} \doteq 1 \rightarrow \langle \mathrm{u.age} = 2; \rangle \mathrm{o.age} \doteq \mathrm{u.age} \qquad \textbf{?}$$

  Depends on whether $I \models \mathrm{o} \doteq \mathrm{u}$

Some Java Features

- Assignments, **complex** expressions with **side effects**
  `int i = 0; if ((i=2) >= 2) {i++;}` `//` value of i?
- Abrupt termination
  - **Exceptions** (`try` – `catch` – `finally`)
  - **Local jumps** `return`, `break`, `continue`
- **Aliasing**
  Different navigation expressions may be same object reference

$$I \models \mathrm{o.age} \doteq 1 \rightarrow \langle \mathrm{u.age} = 2; \rangle \mathrm{o.age} \doteq \mathrm{u.age} \qquad ?$$

  Depends on whether $I \models \mathrm{o} \doteq \mathrm{u}$
- **Method calls**, blocks

Some Java Features

- Assignments, **complex** expressions with **side effects**
  `int i = 0; if ((i=2) >= 2) {i++;}` `//` value of i?
- Abrupt termination
  - **Exceptions** (`try` – `catch` – `finally`)
  - **Local jumps** `return`, `break`, `continue`
- **Aliasing**
  Different navigation expressions may be same object reference

$$I \models o.age \doteq 1 \rightarrow \langle u.age = 2; \rangle o.age \doteq u.age \qquad ?$$

  Depends on whether $I \models o \doteq u$
- **Method calls**, blocks

Solution within KeY to be discussed in detail

More Java Features

Addressed in KeY

- Java's rules for localisation of attributes and method implementations (**polymorphism**, **dynamic binding**)

More Java Features

Addressed in KeY

- Java's rules for localisation of attributes and method implementations (**polymorphism**, **dynamic binding**)
  - Scope (class/instance)
  - Context (static/runtime)
  - Visibility
  - `super`

More Java Features

Addressed in KeY

- Java's rules for localisation of attributes and method implementations (**polymorphism**, **dynamic binding**)
  - Scope (class/instance)
  - Context (static/runtime)
  - Visibility
  - `super`

  Solution: branch proof if implementation not uniquely determined

More Java Features

Addressed in KeY

- Java's rules for localisation of attributes and method implementations (**polymorphism**, **dynamic binding**)
  - Scope (class/instance)
  - Context (static/runtime)
  - Visibility
  - `super`

  Solution: branch proof if implementation not uniquely determined
- Run time errors (`null` **pointer exceptions**)
  Functions that model attributes are partially defined

More Java Features

Addressed in KeY

- Java's rules for localisation of attributes and method implementations (**polymorphism**, **dynamic binding**)
  - Scope (class/instance)
  - Context (static/runtime)
  - Visibility
  - `super`

  Solution: branch proof if implementation not uniquely determined
- Run time errors (`null` **pointer exceptions**)
  Functions that model attributes are partially defined
  Solution: optional rule set enforces proof of $!(o \doteq \texttt{null})$
  (whenever object reference o accessed)

More Java Features

- Java Card data types
  `boolean`, `char`, `String`
  `int`, `byte`, `long` (cyclic!)
  Arrays
  Solution: optional rule sets ***N***/`int`, rules for built-ins

More Java Features

- Java Card data types
  `boolean`, `char`, `String`
  `int`, `byte`, `long` (cyclic!)
  Arrays
  Solution: optional rule sets $\mathbb{N}$/`int`, rules for built-ins
- Object **creation** and **initialisation**
  Trick to keep same universe U in all states:
  all objects exist anytime, use attributes $o.\mathrm{created}$, $o.\mathrm{initialized}$

Side Effects and Complex Expressions

```
int i = 0; if ((i=2) >= 2) {i++;} // value of i?
```

JAVA expressions can assign values (**assignment operators**)
FOL/DL terms have **no** side effects

Side Effects and Complex Expressions

```
int i = 0; if ((i=2) >= 2) {i++;} // value of i?
```

JAVA expressions can assign values (**assignment operators**)
FOL/DL terms have **no** side effects

Decomposition of complex terms following symbolic execution as defined for expressions JAVA language specification
Local **program transformations**

$$\text{ITERATED-ASSIGNMENT}\ \frac{\Gamma \Longrightarrow \langle \pi\ \mathtt{y} = \mathtt{t};\ \mathtt{x} = \mathtt{y};\ \omega \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \pi\ \mathtt{x} = \mathtt{y} = \mathtt{t};\ \omega \rangle \phi, \Delta} \qquad t \text{ simple}$$

Side Effects and Complex Expressions

```
int i = 0; if ((i=2) >= 2) {i++;} // value of i?
```

JAVA expressions can assign values (**assignment operators**)
FOL/DL terms have **no** side effects

Decomposition of complex terms following symbolic execution as defined for expressions JAVA language specification
Local **program transformations**

$$\text{ITERATED-ASSIGNMENT}\ \frac{\Gamma \Longrightarrow \langle \pi\ \mathtt{y} = \mathtt{t};\ \mathtt{x} = \mathtt{y};\ \omega \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \pi\ \mathtt{x} = \mathtt{y} = \mathtt{t};\ \omega \rangle \phi, \Delta} \qquad t \text{ simple}$$

Temporary program variables '_var<n>' store intermediate results

$$\text{IF-EVAL}\ \frac{\Gamma \Longrightarrow \langle \pi\ \mathtt{boolean}\ \mathtt{v_{new}};\ \mathtt{v_{new}} = \mathtt{b};\ \mathtt{if}\ (\mathtt{v_{new}})\ \{\alpha\};\ \omega \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \pi\ \mathtt{if}\ (b)\ \{\alpha\};\ \omega \rangle \phi, \Delta}$$

where b complex

Side Effects and Complex Expressions, Cont'd

Applying rule to statement including guard with side effect is incorrect

Restrict applicability of IF-THEN and other rules with guards:
Guard expression needs to be **simple** (ie, side effect-free)

$$\text{IF-SPLIT}\ \frac{\Gamma, b \doteq \texttt{TRUE} \texttt{ ==> } \langle \pi\ \alpha;\ \omega \rangle \phi, \Delta \qquad \Gamma, b \doteq \texttt{FALSE} \texttt{ ==> } \langle \pi\ \ \omega \rangle \phi, \Delta}{\Gamma \texttt{ ==> } \langle \pi\ \texttt{if}\ (b)\ \{\alpha\};\ \omega \rangle \phi, \Delta}$$

where b simple

Demo

`javaDL/complex.key`

Abrupt Termination

Redirection of control flow via `return`, `break`, `continue`, **exceptions**

$$\langle \pi \texttt{ try } \{\xi\alpha\} \texttt{ catch(e) } \{\gamma\} \texttt{ finally } \{\epsilon\};\ \omega \rangle \phi$$

Abrupt Termination

Redirection of control flow via `return`, `break`, `continue`, **exceptions**

$$\langle \pi \texttt{ try } \{\xi\alpha\} \texttt{ catch(e) } \{\gamma\} \texttt{ finally } \{\epsilon\};\ \omega\rangle\phi$$

Solution: rules work on first active statement, `try` part of prefix

Abrupt Termination

Redirection of control flow via `return`, `break`, `continue`, **exceptions**

$$\langle \pi \texttt{ try } \{\xi\alpha\} \texttt{ catch(e) } \{\gamma\} \texttt{ finally } \{\epsilon\};\ \omega\rangle\phi$$

Solution: rules work on first active statement, `try` part of prefix

TRY-THROW (exc simple)

$$\frac{\Gamma \Longrightarrow \left\langle \begin{array}{l} \pi \texttt{ if (exc instanceOf Exception) \{} \\ \quad \texttt{try \{e = exc; } \gamma\texttt{\} finally \{}\epsilon\texttt{\}} \\ \texttt{\} else \{}\epsilon \texttt{ throw exc\}; } \omega \end{array} \right\rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \pi \texttt{ try \{throw exc; } \alpha\texttt{\} catch(e) \{}\gamma\texttt{\} finally \{}\epsilon\texttt{\}; } \omega\rangle\phi, \Delta}$$

Aliasing

Naive alias resolution causes **proof split** (on $o \doteq u$) at each access

$$\Gamma \texttt{ ==> } o.age \doteq 1 \texttt{ -> } \langle u.age = 2; \rangle o.age \doteq u.age, \Delta$$

Aliasing

Naive alias resolution causes **proof split** (on $o \doteq u$) at each access

$$\Gamma \texttt{ ==> } o.age \doteq 1 \texttt{ -> } \langle u.age = 2; \rangle o.age \doteq u.age, \Delta$$

Unnecessary in many cases!

$$\Gamma \texttt{ ==> } o.age \doteq 1 \texttt{ -> } \langle u.age = 2; \quad o.age = 2; \rangle o.age \doteq u.age, \Delta$$

$$\Gamma \texttt{ ==> } o.age \doteq 1 \texttt{ -> } \langle u.age = 2; \rangle u.age \doteq 2, \Delta$$

Aliasing

Naive alias resolution causes **proof split** (on $o \doteq u$) at each access

$$\Gamma \texttt{ ==> } o.age \doteq 1 \texttt{ -> } \langle u.age = 2; \rangle o.age \doteq u.age, \Delta$$

Unnecessary in many cases!

$$\Gamma \texttt{ ==> } o.age \doteq 1 \texttt{ -> } \langle u.age = 2; \quad o.age = 2; \rangle o.age \doteq u.age, \Delta$$

$$\Gamma \texttt{ ==> } o.age \doteq 1 \texttt{ -> } \langle u.age = 2; \rangle u.age \doteq 2, \Delta$$

Updates avoid such proof splits:

- Delayed state computation until clear what **actually** required
- Simplification of updates

Updates for JAVA

Let *loc* be either one of

- program variable `x`

Updates for JAVA

Let *loc* be either one of

- program variable `x`
- attribute access `o.attr` (`o` has object type)

Updates for JAVA

Let *loc* be either one of

- program variable `x`
- attribute access `o.attr` (`o` has object type)
- array access `a[i]` (`a` has array type, not discussed here)

Updates for JAVA

Let *loc* be either one of

- program variable `x`
- attribute access `o.attr` (`o` has object type)
- array access `a[i]` (`a` has array type, not discussed here)

$$\text{ASSIGN}\ \frac{\Gamma \Longrightarrow \{loc := val\}\langle \pi\ \ \omega\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle \pi\ loc{=}val;\ \omega\rangle\phi, \Delta}$$

where

- *loc* and *val* satisfy above restrictions
- *val* is a side-effect free term,
- $\{loc := val\}$ is DL **update** (usage and semantics as in simple DL)

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\texttt{\textbackslash if (t1 = t2) \textbackslash then t \textbackslash else e})^{I,\beta} = \begin{cases} t^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ e^{I,\beta} & \text{otherwise} \end{cases}$$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\texttt{\textbackslash if (t1 = t2) \textbackslash then t \textbackslash else e})^{I,\beta} = \begin{cases} t^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ e^{I,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

$$\begin{aligned} \{\mathbf{o.a} := t\}\mathbf{o.a} &\rightsquigarrow t \\ \{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{b} &\rightsquigarrow (\{\texttt{o.a} := t\}\texttt{u}).\texttt{b} \\ \{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{a} &\rightsquigarrow \\ &\quad \texttt{\textbackslash if } ((\{\texttt{o.a:=t}\}\texttt{u})\texttt{=o}) \texttt{ \textbackslash then t \textbackslash else } (\{\texttt{o.a:=t}\}\texttt{u}).\texttt{a} \end{aligned}$$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\texttt{\textbackslash if (t1 = t2) \textbackslash then t \textbackslash else e})^{I,\beta} = \begin{cases} t^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ e^{I,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

$$\begin{aligned} \{\mathbf{o.a} := t\}\mathbf{o.a} &\rightsquigarrow t \\ \{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{b} &\rightsquigarrow (\{\texttt{o.a} := t\}\texttt{u}).\texttt{b} \\ \{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{a} &\rightsquigarrow \\ &\quad \texttt{\textbackslash if } ((\{\texttt{o.a:=t}\}\texttt{u})\texttt{=o}) \texttt{ \textbackslash then t \textbackslash else } (\{\texttt{o.a:=t}\}\texttt{u}).\texttt{a} \end{aligned}$$

Example

$\{\texttt{o.a} := \texttt{o}\}\texttt{o.a.a.b}$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\texttt{\textbackslash if (t1 = t2) \textbackslash then t \textbackslash else e})^{I,\beta} = \begin{cases} t^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ e^{I,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

$$\{\mathbf{o.a} := t\}\mathbf{o.a} \rightsquigarrow t$$

$$\{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{b} \rightsquigarrow (\{\texttt{o.a} := t\}\texttt{u}).\texttt{b}$$

$$\{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{a} \rightsquigarrow \texttt{\textbackslash if (}(\{\texttt{o.a:=t}\}\texttt{u})\texttt{=o) \textbackslash then t \textbackslash else (}\{\texttt{o.a:=t}\}\texttt{u)}\texttt{.a}$$

Example

$$\{\texttt{o.a} := \texttt{o}\}\texttt{o.a.a.b} \rightsquigarrow \{\texttt{o.a} := \texttt{o}\}\texttt{o.a.a.b}$$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\texttt{\textbackslash if (t1 = t2) \textbackslash then t \textbackslash else e})^{I,\beta} = \begin{cases} t^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ e^{I,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

$$\{\mathbf{o.a} := t\}\mathbf{o.a} \rightsquigarrow t$$

$$\{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{b} \rightsquigarrow (\{\texttt{o.a} := t\}\texttt{u}).\texttt{b}$$

$$\{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{a} \rightsquigarrow \texttt{\textbackslash if (}(\{\texttt{o.a:=t}\}\texttt{u})\texttt{=o) \textbackslash then t \textbackslash else (}\{\texttt{o.a:=t}\}\texttt{u)}\texttt{.a}$$

Example

$$\{\texttt{o.a} := \texttt{o}\}\texttt{o.a.a.b} \rightsquigarrow (\{\texttt{o.a} := \texttt{o}\}\texttt{o.a.a}).\texttt{b}$$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\texttt{\textbackslash if (t1 = t2) \textbackslash then t \textbackslash else e})^{I,\beta} = \begin{cases} t^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ e^{I,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

$$\begin{aligned} \{\mathbf{o.a} := t\}\mathbf{o.a} &\rightsquigarrow t \\ \{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{b} &\rightsquigarrow (\{\texttt{o.a} := t\}\texttt{u}).\texttt{b} \\ \{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{a} &\rightsquigarrow \\ &\quad \texttt{\textbackslash if } ((\{\texttt{o.a:=t}\}\texttt{u})\texttt{=o}) \texttt{ \textbackslash then t \textbackslash else } (\{\texttt{o.a:=t}\}\texttt{u})\texttt{.a} \end{aligned}$$

Example

$$\{\texttt{o.a} := \texttt{o}\}\texttt{o.a.a.b} \rightsquigarrow \begin{array}{l} (\ \texttt{\textbackslash if } ((\{\texttt{o.a} := \texttt{o}\}\texttt{o.a}) = \texttt{o}) \\ \quad \texttt{\textbackslash then o} \\ \quad \texttt{\textbackslash else } (\{\texttt{o.a} := \texttt{o}\}\texttt{o.a}).\texttt{a }).\texttt{b} \end{array}$$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\texttt{\textbackslash if (t1 = t2) \textbackslash then t \textbackslash else e})^{I,\beta} = \begin{cases} t^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ e^{I,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

$$\begin{aligned} \{\mathbf{o.a} := t\}\mathbf{o.a} &\rightsquigarrow t \\ \{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{b} &\rightsquigarrow (\{\texttt{o.a} := t\}\texttt{u}).\texttt{b} \\ \{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{a} &\rightsquigarrow \\ &\quad \texttt{\textbackslash if } ((\{\texttt{o.a:=t}\}\texttt{u})\texttt{=o}) \texttt{ \textbackslash then t \textbackslash else } (\{\texttt{o.a:=t}\}\texttt{u}).\texttt{a} \end{aligned}$$

Example

$$\{\texttt{o.a} := \texttt{o}\}\texttt{o.a.a.b} \rightsquigarrow \begin{array}{l} (\ \texttt{\textbackslash if } (\texttt{o} = \texttt{o}) \\ \quad \texttt{\textbackslash then o} \\ \quad \texttt{\textbackslash else o.a }).\texttt{b} \end{array}$$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\texttt{\textbackslash if (t1 = t2) \textbackslash then t \textbackslash else e})^{I,\beta} = \begin{cases} t^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ e^{I,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

$$\{\mathbf{o.a} := t\}\mathbf{o.a} \rightsquigarrow t$$

$$\{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{b} \rightsquigarrow (\{\texttt{o.a} := t\}\texttt{u}).\texttt{b}$$

$$\{\texttt{o.}\mathbf{a} := t\}\texttt{u.}\mathbf{a} \rightsquigarrow \texttt{\textbackslash if (}(\{\texttt{o.a:=t}\}\texttt{u})\texttt{=o) \textbackslash then t \textbackslash else (}\{\texttt{o.a:=t}\}\texttt{u)}\texttt{.a}$$

Example

$$\{\texttt{o.a} := \texttt{o}\}\texttt{o.a.a.b} \rightsquigarrow \texttt{o.b}$$

Parallel Updates

Computing update followed by **update**

$$\{l_1 := r_1\}\{l_2 := r_2\} \quad = \quad \{\{l_1 := r_1\},\ \{\{l_1 := r_1\} \downarrow l_2 := \{l_1 := r_1\}r_2\}\}$$

Results in **parallel update**

Parallel Updates

Computing update followed by **update**

$$\{l_1 := r_1\}\{l_2 := r_2\} \quad = \quad \{\{l_1 := r_1\},\ \{\{l_1 := r_1\} \downarrow l_2 := \{l_1 := r_1\}r_2\}\}$$

Results in **parallel update**

Syntax

$$\{l_1 := v_1,\ \ldots,\ l_n := v_n\}$$

Parallel Updates

Computing update followed by **update**

$$\{l_1 := r_1\}\{l_2 := r_2\} \quad = \quad \{\{l_1 := r_1\},\ \{\{l_1 := r_1\} \downarrow l_2 := \{l_1 := r_1\}r_2\}\}$$

Results in **parallel update**

Syntax

$$\{l_1 := v_1,\ \ldots,\ l_n := v_n\}$$

Semantics

- All l_i and v_i computed in old state
- All updates done simultaneously
- If conflict $\quad l_i = l_j,\ v_i \neq v_j \quad$ later update wins

Method Call

Method call with actual parameters $arg_1, \ldots, arg_n$

$$\{arg_1 := t_1, \ldots, arg_n := t_n,\ o := t_o\}\langle o.m(arg_1, \ldots, arg_n);\rangle\phi$$

Where method declaration is: `void` $m(T_1\ p_1, \ldots, T_n\ p_n)$

Method Call

Method call with actual parameters $arg_1, \ldots, arg_n$

$$\{arg_1 := t_1, \ldots, arg_n := t_n,\ o := t_o\}\langle o.m(arg_1, \ldots, arg_n);\rangle\phi$$

Where method declaration is: `void` $m(T_1\ p_1, \ldots, T_n\ p_n)$

What the rule **method-call** does:

- (type conformance of arg_i to T_i guaranteed by JAVA compiler)

Method Call

Method call with actual parameters $arg_1, \ldots, arg_n$

$$\{arg_1 := t_1, \ldots, arg_n := t_n,\ o := t_o\}\langle o.m(arg_1, \ldots, arg_n);\rangle\phi$$

Where method declaration is: `void` $m(T_1\ p_1, \ldots, T_n\ p_n)$

What the rule **method-call** does:

- (type conformance of arg_i to T_i guaranteed by JAVA compiler)
- for each formal parameter p_i of m declare & initialize new local variable '$T_i\ p_i = arg_i$;'

Method Call

Method call with actual parameters $arg_1, \ldots, arg_n$

$$\{arg_1 := t_1, \ldots, arg_n := t_n,\ o := t_o\}\langle o.m(arg_1, \ldots, arg_n);\rangle\phi$$

Where method declaration is: `void` $m(T_1\ p_1, \ldots, T_n\ p_n)$

What the rule **method-call** does:

- (type conformance of arg_i to T_i guaranteed by JAVA compiler)
- for each formal parameter p_i of m declare & initialize new local variable '$T_i\ p_i = arg_i;$'
- look up implementation class C of m split proof, if implementation not determinable

Method Call

Method call with actual parameters $arg_1, \ldots, arg_n$

$$\{arg_1 := t_1, \ldots, arg_n := t_n,\ o := t_o\}\langle o.m(arg_1, \ldots, arg_n);\rangle\phi$$

Where method declaration is: `void` $m(T_1\ p_1, \ldots, T_n\ p_n)$

What the rule **method-call** does:

- (type conformance of arg_i to T_i guaranteed by JAVA compiler)
- for each formal parameter p_i of m declare & initialize new local variable '$T_i\ p_i = arg_i$;'
- look up implementation class C of m split proof, if implementation not determinable
- make concrete call o.C::$m(p_1, \ldots, p_n)$

Method Body Expand

After processing code that binds actual to formal parameters (symbolic execution of '$T_i\ p_i = arg_i$;')

$$\text{METHOD-BODY-EXPAND}\ \frac{\Gamma \Longrightarrow \langle \pi\ \texttt{method-frame}(C(\mathsf{o})\{\ b\ \}\ \omega \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \pi\ \mathsf{o}.C::m(p_1, \ldots, p_n);\ \omega \rangle \phi, \Delta}$$

Method Body Expand

After processing code that binds actual to formal parameters (symbolic execution of '$T_i\ p_i = arg_i$;')

$$\text{METHOD-BODY-EXPAND}\ \frac{\Gamma \Longrightarrow \langle \pi\ \texttt{method-frame(}C(\mathrm{o})\texttt{)\{}\ b\ \texttt{\}}\ \omega \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \pi\ \mathrm{o}.C::m(p_1, \ldots, p_n);\ \omega \rangle \phi, \Delta}$$

Symbolic Execution

Method Body Expand

After processing code that binds actual to formal parameters (symbolic execution of '$T_i\ p_i = arg_i$;')

$$\text{METHOD-BODY-EXPAND}\ \frac{\Gamma \Longrightarrow \langle \pi\ \texttt{method-frame(}C(\mathsf{o})\texttt{)\{}\ b\ \texttt{\}}\ \omega \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \pi\ \mathsf{o}.C::m(p_1, \ldots, p_n);\ \omega \rangle \phi, \Delta}$$

Symbolic Execution
Only static information available, proof splitting

Method Body Expand

After processing code that binds actual to formal parameters (symbolic execution of '$T_i\ p_i = arg_i$;')

$$\text{METHOD-BODY-EXPAND}\ \frac{\Gamma \Longrightarrow \langle \pi\ \texttt{method-frame(}C(\mathsf{o})\texttt{)\{}\ b\ \texttt{\}}\ \omega \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \pi\ \mathsf{o}.C::m(p_1, \ldots, p_n);\ \omega \rangle \phi, \Delta}$$

Symbolic Execution
Only static information available, proof splitting
Runtime infrastructure required in calculus