

Introduction to JavaCard Dynamic Logic

Prof. P. H. Schmitt, Christian Engel, Benjamin Weiß

December 10, 2008

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

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

Depends on whether $I \models o \doteq u$

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

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

Depends on whether $I \models o \doteq u$

- **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

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

Depends on whether $I \models o \doteq u$

- **Method calls**, blocks

Solution within KeY to be discussed in detail

Addressed in KeY

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

Addressed in KeY

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

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

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

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 $!(o \doteq \text{null})$
(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 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 $o.created$, $o.initialized$

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**

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

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**

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

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

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

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)

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

where b simple

Demo

javaDL/complex.key



Abrupt Termination

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

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


Abrupt Termination

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

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

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



Abrupt Termination

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

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

Solution: rules work on first active statement, **try** part of prefix
TRY-THROW (exc simple)

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

Aliasing

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

$$\Gamma \implies o.\text{age} \doteq 1 \rightarrow \langle u.\text{age} = 2; \rangle o.\text{age} \doteq u.\text{age}, \Delta$$

Aliasing

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

$$\Gamma \implies o.\text{age} \doteq 1 \rightarrow \langle u.\text{age} = 2; \rangle o.\text{age} \doteq u.\text{age}, \Delta$$

Unnecessary in many cases!

$$\Gamma \implies o.\text{age} \doteq 1 \rightarrow \langle u.\text{age} = 2; \text{ o.age} = 2; \rangle o.\text{age} \doteq u.\text{age}, \Delta$$

$$\Gamma \implies o.\text{age} \doteq 1 \rightarrow \langle u.\text{age} = 2; \rangle u.\text{age} \doteq 2, \Delta$$



Aliasing

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

$$\Gamma \implies o.\text{age} \doteq 1 \rightarrow \langle u.\text{age} = 2; \rangle o.\text{age} \doteq u.\text{age}, \Delta$$

Unnecessary in many cases!

$$\Gamma \implies o.\text{age} \doteq 1 \rightarrow \langle u.\text{age} = 2; \quad o.\text{age} = 2; \rangle o.\text{age} \doteq u.\text{age}, \Delta$$

$$\Gamma \implies o.\text{age} \doteq 1 \rightarrow \langle u.\text{age} = 2; \rangle u.\text{age} \doteq 2, \Delta$$

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)

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

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

$$(\text{\if } (t_1 = t_2) \text{\then } t \text{\else } e) \text{\superscript } l, \beta = \begin{cases} t \text{\superscript } l, \beta & t_1 \text{\superscript } l, \beta = t_2 \text{\superscript } l, \beta \\ e \text{\superscript } l, \beta & \text{otherwise} \end{cases}$$



Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\backslash\text{if } (t_1 = t_2) \backslash\text{then } t \backslash\text{else } e)^{l,\beta} = \begin{cases} t^{l,\beta} & t_1^{l,\beta} = t_2^{l,\beta} \\ e^{l,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

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

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

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

$$\backslash\text{if } ((\{\mathbf{o.a} := t\}\mathbf{u}) = \mathbf{o}) \backslash\text{then } t \backslash\text{else } (\{\mathbf{o.a} := t\}\mathbf{u}).\mathbf{a}$$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\backslash\text{if } (t_1 = t_2) \backslash\text{then } t \backslash\text{else } e)^{l,\beta} = \begin{cases} t^{l,\beta} & t_1^{l,\beta} = t_2^{l,\beta} \\ e^{l,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

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

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

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

$$\backslash\text{if } ((\{o.a := t\}u)=o) \backslash\text{then } t \backslash\text{else } (\{o.a := t\}u).a$$

Example

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

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\backslash\text{if } (t_1 = t_2) \backslash\text{then } t \backslash\text{else } e)^{l,\beta} = \begin{cases} t^{l,\beta} & t_1^{l,\beta} = t_2^{l,\beta} \\ e^{l,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

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

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

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

$$\backslash\text{if } ((\{\mathbf{o.a} := t\}\mathbf{u}) = \mathbf{o}) \backslash\text{then } t \backslash\text{else } (\{\mathbf{o.a} := t\}\mathbf{u}).\mathbf{a}$$

Example

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

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\backslash\text{if } (t_1 = t_2) \backslash\text{then } t \backslash\text{else } e)^{l,\beta} = \begin{cases} t^{l,\beta} & t_1^{l,\beta} = t_2^{l,\beta} \\ e^{l,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

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

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

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

$$\backslash\text{if } ((\{o.a := t\}u)=o) \backslash\text{then } t \backslash\text{else } (\{o.a := t\}u).a$$

Example

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

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\backslash\text{if } (t_1 = t_2) \backslash\text{then } t \backslash\text{else } e)^{l,\beta} = \begin{cases} t^{l,\beta} & t_1^{l,\beta} = t_2^{l,\beta} \\ e^{l,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

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

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

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

$$\backslash\text{if } ((\{o.a := t\}u) = o) \backslash\text{then } t \backslash\text{else } (\{o.a := t\}u).a$$

Example

$$\{o.a := o\}o.a.a.b \rightsquigarrow \begin{aligned} & (\backslash\text{if } ((\{o.a := o\}o.a) = o) \\ & \backslash\text{then } o \\ & \backslash\text{else } (\{o.a := o\}o.a).a).b \end{aligned}$$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\backslash\text{if } (t_1 = t_2) \backslash\text{then } t \backslash\text{else } e)^{l,\beta} = \begin{cases} t^{l,\beta} & t_1^{l,\beta} = t_2^{l,\beta} \\ e^{l,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

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

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

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

$$\backslash\text{if } ((\{\mathbf{o.a} := t\}\mathbf{u}) = \mathbf{o}) \backslash\text{then } t \backslash\text{else } (\{\mathbf{o.a} := t\}\mathbf{u}).\mathbf{a}$$

Example

$$\{\mathbf{o.a} := \mathbf{o}\}\mathbf{o.a.a.b} \rightsquigarrow \begin{aligned} &(\backslash\text{if } (\mathbf{o} = \mathbf{o}) \\ &\backslash\text{then } \mathbf{o} \\ &\backslash\text{else } \mathbf{o.a}).\mathbf{b} \end{aligned}$$

Computing Effect of Updates: Attributes

Use **conditional terms** to delay splitting further

$$(\backslash\text{if } (t_1 = t_2) \backslash\text{then } t \backslash\text{else } e)^{l,\beta} = \begin{cases} t^{l,\beta} & t_1^{l,\beta} = t_2^{l,\beta} \\ e^{l,\beta} & \text{otherwise} \end{cases}$$

Computing update followed by **attribute access**

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

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

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

$$\backslash\text{if } ((\{o.a := t\}u)=o) \backslash\text{then } t \backslash\text{else } (\{o.a := t\}u).a$$

Example

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

Parallel Updates

Computing update followed by **update** results in **parallel update**:

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

$$\text{where } \{l := r\} \downarrow f(t_1, \dots, t_n) = f(\{l := r\}t_1, \dots, \{l := r\}t_n)$$



Parallel Updates

Computing update followed by **update** results in **parallel 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 $\{l := r\} \downarrow f(t_1, \dots, t_n) = f(\{l := r\}t_1, \dots, \{l := r\}t_n)$

Syntax

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

Parallel Updates

Computing update followed by **update** results in **parallel 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 $\{l := r\} \downarrow f(t_1, \dots, t_n) = f(\{l := r\}t_1, \dots, \{l := r\}t_n)$

Syntax

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

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

Method Call

Method call with actual parameters arg_1, \dots, arg_n

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

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



Method Call

Method call with actual parameters arg_1, \dots, arg_n

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

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

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, \dots, arg_n

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

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

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, \dots, arg_n

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

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

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, \dots, arg_n

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

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

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 $o.C::m(p_1, \dots, p_n)$



Method Body Expand

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

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

Method Body Expand

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

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

Symbolic Execution

Method Body Expand

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

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

Symbolic Execution

Only static information available, proof splitting



Method Body Expand

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

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

Symbolic Execution

Only static information available, proof splitting

Runtime infrastructure required in calculus

