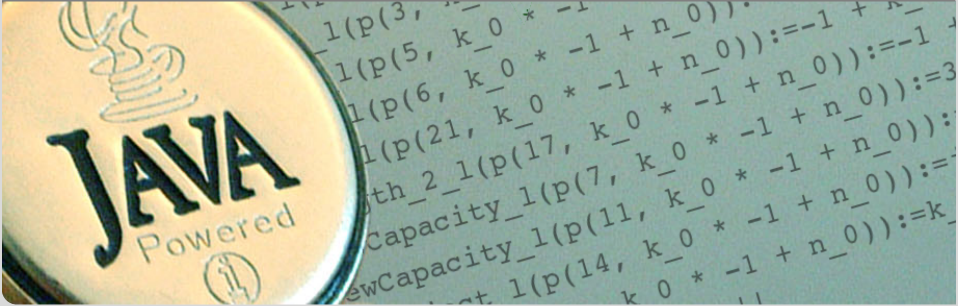
Karlsruhe Institute of Technology

Specification & Formal Analysis of Java Programs

Functional Verification of Java Programs

Prof. Dr. Bernhard Beckert | ADAPT 2010

KIT – INSTITUT FÜR THEORETISCHE INFORMATIK



Dynamic Logic Formulas (Simple Version)

Definition (Dynamic Logic Formulas (DL Formulas))

- Each FOL formula is a DL formula
- If p is a program and ϕ a DL formula then $\left\{\begin{matrix}\langle \mathrm{p} \rangle \phi \\ [\mathrm{p}]\phi\end{matrix}\right\}$ is a DL formula
- DL formulas closed under FOL quantifiers and connectives

- Program variables are flexible *constants*: never bound in quantifiers
- Program variables need not be declared or initialized in program
- Programs contain no logical variables
- Modalities can be arbitrarily nested

Dynamic Logic Formulas Cont'd

Example (Well-formed? If yes, under which signature?)

- $\forall$ **int** y; $((\langle$`x = 1;`$\rangle$`x` $\doteq y) <-> (\langle$`x = 1*1;`$\rangle$`x` $\doteq y))$

  Well-formed if $FSym_{nr}$ contains **int** `x;`
- $\exists$ **int** x; [`x = 1;`](`x` $\doteq 1$)

  Not well-formed, because logical variable occurs in program
- $\langle$`x = 1;`$\rangle$([**while** (**true**) {};]**false**)

  Well-formed if $FSym_{nr}$ contains **int** `x;`
  program formulas can be nested

Semantic Evaluation of Program Formulas

Definition (Validity Relation for Program Formulas)

- $s, \beta \models \langle \mathtt{p} \rangle \phi$ iff $\rho(\mathtt{p})(s), \beta \models \phi$ *and* $\rho(\mathtt{p})(s)$ *is defined*

  $\mathtt{p}$ *terminates* and ϕ is true in the final state after execution
- $s, \beta \models [\mathtt{p}] \phi$ iff $\rho(\mathtt{p})(s), \beta \models \phi$ *whenever* $\rho(\mathtt{p})(s)$ *is defined*

  If $\mathtt{p}$ *terminates then* ϕ is true in the final state after execution

Program Correctness

Definition (Notions of Correctness)

- If $s, \beta \models \langle \mathrm{p} \rangle \phi$ then

  $\mathtt{p}$ *totally correct* (with respect to ϕ) in s, β
- If $s, \beta \models [\mathrm{p}]\phi$ then
  $\mathtt{p}$ *partially correct* (with respect to ϕ) in s, β

- *Duality* $\langle \mathrm{p} \rangle \phi$ iff $![\mathrm{p}]!\,\phi$
  Exercise: justify this with help of semantic definitions
- *Implication* if $\langle \mathrm{p} \rangle \phi$ then $[\mathrm{p}]\phi$
  Total correctness implies partial correctness
  - converse is false
  - holds only for deterministic programs

Semantics of Sequents

$\Gamma = \{\phi_1, \ldots, \phi_n\}$ and $\Delta = \{\psi_1, \ldots, \psi_m\}$ sets of program formulas
where all logical variables occur bound

Recall: $s \models (\Gamma \Longrightarrow \Delta)$ iff $s \models (\phi_1 \ \& \cdots \& \ \phi_n) \ \texttt{->} \ (\psi_1 \mid \cdots \mid \psi_m)$

Define semantics of DL sequents identical to semantics of FOL sequents

Definition (Validity of Sequents over Program Formulas)

A sequent $\Gamma \Longrightarrow \Delta$ over program formulas is *valid* iff

$$s \models (\Gamma \Longrightarrow \Delta) \text{ in } \textit{all states } s$$

Consequence for program variables

Initial value of program variables implicitly “universally quantified”

Initial States

Java initial states

KeY prover "starts" programs in initial states according to Java convention:

- Values of array entries initialized to default values: `int[]` to 0, etc.
- Static object initialization
- No objects created

How to restrict validity to set of *initial states* $S_0 \subseteq S$?

1. Design closed FOL formula Init with
   $s \models \mathrm{Init} \quad \text{iff} \quad s \in S_0$
2. Use sequent $\Gamma, \mathrm{Init} \Longrightarrow \Delta$

Operational Semantics of Programs

In labelled transition system $K = (S, \rho)$:
$\rho : \Pi \to (S \to S)$ is *operational semantics* of programs $\mathtt{p} \in \Pi$

How is ρ defined for concrete programs and states?

Example (Operational semantics of assignment)

States s interpret non-rigid symbols f with $\mathcal{I}_s(f)$

$\rho(\mathtt{x=t})(s) = s'$ where s' identical to s except $\mathcal{I}_{s'}(x) = val_s(t)$

Very tedious task to define ρ for Java ...
$\Rightarrow$ go directly to calculus for program formulas!

Symbolic Execution of Programs

Sequent calculus decomposes top-level operator in formula
What is “top-level” in a sequential program `p; q; r` ?

Symbolic Execution (King, late 60s)

- Follow the *natural control flow* when analysing a program
- Values of some variables unknown: *symbolic state representation*

Example

Compute the final state after termination of

```
int x; int y; x=x+y; y=x-y; x=x-y;
```

Symbolic Execution of Programs Cont'd

General form of rule conclusions in symbolic execution calculus

$$\langle \texttt{stmt; rest} \rangle \phi, \qquad [\texttt{stmt; rest}]\phi$$

- Rules must *symbolically execute* first statement
- Repeated application of rules in a proof corresponds to *symbolic program execution*

Symbolic Execution of Programs Cont'd

Symbolic execution of assignment

$$\text{assign}\ \frac{\{\mathtt{x}/\mathtt{x}_{old}\}\Gamma,\ \mathtt{x} \doteq \{\mathtt{x}/\mathtt{x}_{old}\}t \quad\Longrightarrow\quad \langle\mathtt{rest}\rangle\phi,\ \{\mathtt{x}/\mathtt{x}_{old}\}\Delta}{\Gamma \Longrightarrow \langle\mathtt{x\ =\ t;\ \ rest}\rangle\phi, \Delta}$$

$\mathtt{x}_{old}$ new program variable that "rescues" old value of $\mathtt{x}$

Example

Conclusion matching: $\{\mathtt{x}/\mathtt{x}\}$, $\{\mathtt{t}/\mathtt{x+y}\}$, $\{\mathtt{rest}/\mathtt{y=x-y;\ \ x=x-y;}\}$, $\{\phi/(\mathtt{x} \doteq y_0\ \&\ \mathtt{y} \doteq x_0)\}$, $\{\Gamma/\mathtt{x} \doteq x_0, \mathtt{y} \doteq y_0\}$, $\{\Delta/\emptyset\}$

$$\frac{\mathtt{x}_{old} \doteq x_0,\ \mathtt{y} \doteq y_0,\ \mathtt{x} \doteq \mathtt{x}_{old}+\mathtt{y} \Longrightarrow \langle\mathtt{y=x-y;\ \ x=x-y;}\rangle(\mathtt{x} \doteq y_0\ \&\ \mathtt{y} \doteq x_0)}{\mathtt{x} \doteq x_0,\ \mathtt{y} \doteq y_0 \Longrightarrow \langle\mathtt{x=x+y;\ \ y=x-y;\ \ x=x-y;}\rangle(\mathtt{x} \doteq y_0\ \&\ \mathtt{y} \doteq x_0)}$$

Proving Partial Correctness

Partial correctness assertion

If program p is started in a state satisfying Pre and terminates, then its final state satisfies Post

In Hoare logic {Pre} p {Post} (Pre, Post must be FOL)

In DL Pre -> [p]Post (Pre, Post any DL formula)

Example (In KeY Syntax, Demo automatic proof)

```
\programVariables {
  int x; int y; }

\problem {
  (\forall int x0; \forall int y0; ((x=x0 & y=y0) ->
     \<{x=x+y; y=x-y; x=x-y;}\>(x=y0 & y=x0)))
}
```

More Properties

Example

$\forall\, T\; y;\; ((\langle \mathrm{p} \rangle \mathrm{x} \doteq y) \;\; \texttt{<->} \;\; (\langle \mathrm{q} \rangle \mathrm{x} \doteq y))$

Not valid in general

Programs p behave q equivalently on variable *T* x

Example

$\exists\, T\; y;\; (\mathrm{x} \doteq y \;\; \texttt{->} \;\; \langle \mathrm{p} \rangle \texttt{\textbf{true}})$

Not valid in general

Program p terminates in all states where x has suitable initial value

Symbolic Execution of Programs Cont'd

Symbolic execution of conditional

$$\text{if}\ \frac{\Gamma, \mathtt{b} \doteq \mathbf{true} \Longrightarrow \langle \mathtt{p;\ rest} \rangle \phi, \Delta \qquad \Gamma, \mathtt{b} \doteq \mathbf{false} \Longrightarrow \langle \mathtt{q;\ rest} \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \mathbf{if}\ \mathtt{(b)\ \{\ p\ \}}\ \mathbf{else}\ \mathtt{\{\ q\ \}\ ;\ rest} \rangle \phi, \Delta}$$

Symbolic execution must consider all possible execution branches

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \langle \mathbf{if}\ \mathtt{(b)\ \{\ p;}\ \mathbf{while}\ \mathtt{(b)\ p\};\ r} \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \mathbf{while}\ \mathtt{(b)\ \{p\};\ r} \rangle \phi, \Delta}$$

Quantifying over Program Variables

How to express correctness for any initial value of program variable?

Not allowed: $\forall\, T\ \mathtt{i};\ \langle \ldots \mathtt{i} \ldots \rangle \phi$
(program $\neq$ logical variable)

Not intended: $\Longrightarrow \langle \ldots \mathtt{i} \ldots \rangle \phi$ (Validity of sequents: quantification over *all* states)

As previous: $\forall\, T\ i_0;\ (i_0 \doteq \mathtt{i}\ \texttt{->}\ \langle \ldots \mathtt{i} \ldots \rangle \phi)$

Solution

Use explicit construct to record values in *current* state

Update $\forall\, T\ i_0;\ (\{\mathtt{i} := i_0\} \langle \ldots \mathtt{i} \ldots \rangle \phi)$

Explicit State Updates

Updates specify computation state where formula is evaluated

Definition (Syntax of Updates)

If $\mathtt{v}$ is program variable, t FOL term type-compatible with $\mathtt{v}$, t' any FOL term, and ϕ any DL formula, then

- $\{\mathtt{v} := t\}t'$ is DL term
- $\{\mathtt{v} := t\}\phi$ is DL formula

Definition (Semantics of Updates)

State s interprets non-rigid symbols f with $\mathcal{I}_s(f)$
β variable assignment for logical variables in t

$\rho(\{\mathtt{v} := t\})(s) = s'$ where s' identical to s except $\mathcal{I}_{s'}(x) = val_{s,\beta}(t)$

Explicit State Updates Cont'd

Facts about updates $\{\mathtt{v} := t\}$

- Update semantics identical to assignment
- Value of update depends on logical variables in t:
- Updates as "lazy" assignments (no term substitution done)
- Updates are *not assignments*: right-hand side is FOL term

  $\{\mathtt{x} := n\}\phi$ cannot be turned into assignment (n logical variable)

  $\langle\mathtt{x=i++;}\rangle\phi$ cannot directly be turned into update
- Updates are *not equations*: change value of non-rigid terms
- KeY simplifies and applies (if possible) updates automatically.

Assignment Rule Using Updates

Symbolic execution of assignment using updates

$$\text{assign}\ \frac{\Gamma \Longrightarrow \{\mathtt{x} := t\}\langle\mathtt{rest}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle\mathtt{x\ =\ t;\ \ rest}\rangle\phi, \Delta}$$

- Avoids renaming of program variables
- Works as long as *t* has no side effects (ok in simple DL)
- Special cases for $\mathtt{x} = t_1 + t_2$, etc.

Demo

`swap.key`

Example Proof

Example

```
\programVariables {
  int x;
}
\problem {
  (\exists int y;
    ({x := y}\<{while (x > 0) {x = x-1;}}\> x=0 ))
}
```

Intuitive Meaning? Satisfiable? Valid?

Demo

term.key

What to do when we *cannot* determine a concrete loop bound?

Parallel Updates

How to apply updates on updates?

Example

Symbolic execution of

```
int x; int y; x=x+y; y=x-y; x=x-y;
```

yields:

```
{x := x+y}{y := x-y}{x := x-y}
```

Need to compose three sequential state changes into a single one!

Parallel Updates Cont'd

Karlsruhe Institute of Technology

Definition (Parallel Update)

A *parallel update* is expression of the form
$\{l_1 := v_1 || \cdots || l_n := v_n\}$ where each $\{l_i := v_i\}$ is simple update

- All v_i computed in old state before update is applied
- Updates of all locations l_i executed simultaneously
- Upon *conflict* $l_i = l_j, v_i \neq v_j$ later update ($\max\{i, j\}$) wins

Definition (Composition Sequential Updates/Conflict Resolution)

$$\{l_1 := r_1\}\{l_2 := r_2\} = \{l_1 := r_1 || l_2 := \{l_1 := r_1\} r_2\}$$

$$\{l_1 := v_1 || \cdots || l_n := v_n\}\mathrm{x} = \begin{cases} \mathrm{x} & \text{if } \mathrm{x} \notin \{l_1, \ldots, l_n\} \\ v_k & \text{if } \mathrm{x} = l_k,\ \mathrm{x} \notin \{l_{k+1}, \ldots, l_n\} \end{cases}$$

Parallel Updates Cont'd

Example

```
({x := x+y}{y := x-y}){x := x-y} =
{x := x+y || y := (x+y)-y}{x := x-y} =
{x := x+y || y := (x+y)-y || x := (x+y)-((x+y)-y)} =
{x := x+y || y := x || x := y} =
{y := x || x := y}
```

KeY automatically deletes overwritten (unnecessary) updates

Demo

`swap.key`

Parallel updates to store intermediate state of symbolic computation

A Warning

First-order rules that substitute arbitrary terms

$$\exists\text{-right}\ \frac{\Gamma \Longrightarrow [x/t']\,\phi,\ \exists\, T\, x;\ \phi, \Delta}{\Gamma \Longrightarrow \exists\, T\, x;\ \phi, \Delta} \qquad \forall\text{-left}\ \frac{\Gamma, \forall\, T\, x;\ \phi,\ [x/t']\,\phi \Longrightarrow \Delta}{\Gamma, \forall\, T\, x;\ \phi \Longrightarrow \Delta}$$

$$\text{applyEq}\ \frac{\Gamma, t \doteq t', [t/t']\,\psi \Longrightarrow [t/t']\,\phi, \Delta}{\Gamma, t \doteq t', \psi \Longrightarrow \phi, \Delta}$$

t, t' must be *rigid*, because all occurrences must have the same value

Example

$$\frac{\Gamma, \mathtt{i} \doteq 0 \ \texttt{->}\ \langle\mathtt{i++}\rangle\mathtt{i} \doteq 0 \Longrightarrow \Delta}{\Gamma, \forall\, T\, x;\ (x \doteq 0 \ \texttt{->}\ \langle\mathtt{i++}\rangle x \doteq 0) \Longrightarrow \Delta}$$

Logically valid formula would result in unsatisfiable antecedent!
KeY prohibits unsound substitutions