Formal Methods for Java

Lecture 24: Proving Loops with KeY

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

January 29, 2013

The KeY-Project

- Theorem Prover
- Developed at University of Karlsruhe
- `http://www.key-project.org/`.

- Theory specialized for Java(Card).
- Can generate proof-obligations from JML specification.
- Underlying theory: Sequent Calculus + Dynamic Logic

Rigid vs.Non-Rigid Functions vs. Variables

KeY distinguishes the following symbols:

- Rigid Functions: These are functions that do not depend on the current state of the program.
  - $+, -, *$: *integer* $\times$ *integers* $\to$ *integer* (mathematical operations)
  - $0, 1, \dots$: *integer*, *TRUE*, *FALSE* : *boolean* (mathematical constants)
- Non-Rigid Functions: These are functions that depend on current state.
  - $\cdot[\cdot] : \top \times \mathit{int} \to \top$ (array access)
  - $\texttt{.next} : \top \to \top$ if `next` is a field of a class.
  - $\texttt{i}, \texttt{j} : \top$ if `i`, `j` are program variables.
- Variables: These are logical variables that can be quantified. Variables may not appear in programs.
  - x, y, z

Example

$$\forall x . \mathtt{i} = x \rightarrow \langle \{ \mathit{while}(\mathtt{i} > 0)\{\mathtt{i} = \mathtt{i} - 1;\}\} \rangle \mathtt{i} = 0$$

- $0, 1, -$ are rigid functions.
- $>$ is a rigid relation.
- $\mathtt{i}$ is a non-rigid function.
- x is a logical variable.

Quantification over $\mathtt{i}$ is not allowed and x must not appear in a program.

Builtin Rigid Functions

- $+, -, *, /, \%, \mathit{jdiv}, \mathit{jmod}$: operations on *integer*.
- $\ldots, -1, 0, 1, \ldots$, *TRUE*,*FALSE*, *null*: constants.
- (A) for any type A: cast function.
- $A :: \mathit{get}$ gives the n-th object of type A.

Updates in KeY

The formula $\langle \texttt{i} = t; \alpha \rangle \phi$ is rewritten to

$$\{\texttt{i} := t\}\langle \alpha \rangle \phi$$

Formula $\{\texttt{i} := t\}\phi$ is true, iff
ϕ holds in a state, where the program variable $\texttt{i}$ has the value denoted by the term t.
Here:

- $\texttt{i}$ is a program variable (non-rigid function).
- t is a term (may contain logical variables).
- ϕ a formula

Simplifying Updates

If ϕ contains no modalities, then $\{x := t\}\phi$ is rewritten to $\phi[t/x]$.

A double update $\{x_1 := t_1, x_2 := t_2\}\{x_1 := t_1', x_3 := t_3'\}\phi$ is automatically rewritten to

$$\{x_1 := t_1'[t_1/x_1, t_2/x_2], x_2 := t_2, x_3 := t_3'[t_1/x_1, t_2/x_2]\}\phi$$

Example: $\langle\{\mathtt{i} = \mathtt{j}; \mathtt{j} = \mathtt{i} + 1\}\rangle \mathtt{i} = \mathtt{j}$

$$
\begin{aligned}
&\langle\{\mathtt{i} = \mathtt{j}; \mathtt{j} = \mathtt{i} + 1\}\rangle \mathtt{i} = \mathtt{j} \\
\equiv&\{\mathtt{i} := \mathtt{j}\}\{\mathtt{j} := \mathtt{i+1}\}\mathtt{i} = \mathtt{j} \\
\equiv&\{\mathtt{i} := \mathtt{j}, \mathtt{j} := \mathtt{j} + 1\}\mathtt{i} = \mathtt{j} \\
\equiv&\mathtt{j} = \mathtt{j} + 1 \\
\equiv&\textbf{false}
\end{aligned}
$$

or alternatively

$$
\begin{aligned}
&\langle\{\mathtt{i} = \mathtt{j}; \mathtt{j} = \mathtt{i} + 1\}\rangle \mathtt{i} = \mathtt{j} \\
\equiv&\{\mathtt{i} := \mathtt{j}\}\{\mathtt{j} := \mathtt{i+1}\}\mathtt{i} = \mathtt{j} \\
\equiv&\{\mathtt{i} := \mathtt{j}\}\mathtt{i} = \mathtt{i} + 1 \\
\equiv&\mathtt{j} = \mathtt{j} + 1 \\
\equiv&\textbf{false}
\end{aligned}
$$

Rules for Java Dynamic Logic

- $\langle\{\mathtt{i} = \mathtt{j}; \ldots\}\rangle\phi$ is rewritten to:
  $\{\mathtt{i} := \mathtt{j}\}\langle\{\ldots\}\rangle\phi$.
- $\langle\{\mathtt{i} = \mathtt{j} + \mathtt{k}; \ldots\}\rangle\phi$ is rewritten to:
  $\{\mathtt{i} := \mathtt{j} + \mathtt{k}\}\langle\{\ldots\}\rangle\phi$.
- $\langle\{\mathtt{i} = \mathtt{j} + +; \ldots\}\rangle\phi$ is rewritten to:
  $\langle\{\textbf{int } \mathtt{j\_0}; \mathtt{j\_0} = \mathtt{j}; \mathtt{j} = \mathtt{j} + 1; \mathtt{i} = \mathtt{j\_0}; \ldots\}\rangle\phi$.
- $\langle\{\textbf{int } \mathtt{k}; \ldots\}\rangle\phi$ is rewritten to:
  $\langle\{\ldots\}\rangle\phi$ and $\mathtt{k}$ is added as new program variable.

Rules for Java Dynamic Logic (if statements)

- $\langle\{\text{if } (i < j) s_1 \text{ else } s_2; \ldots\}\rangle\phi$ is rewritten to:
  $\backslash\text{if } i < j \ \backslash\text{then } \langle\{s_1\}; \ldots\rangle\phi \ \backslash\text{else } \langle\{s_2\}; \ldots\rangle\phi$.
- \if ... \then ... \else ... is a logical operator with the following sequent calculus rules:

$$\frac{\Gamma, \phi, \psi_1 \Longrightarrow \Delta \quad \Gamma, \psi_2 \Longrightarrow \phi, \Delta}{\Gamma, \backslash\text{if } \phi \ \backslash\text{then } \psi_1 \ \backslash\text{else } \psi_2 \Longrightarrow \Delta} \qquad \frac{\Gamma, \phi \Longrightarrow \psi_1, \Delta \quad \Gamma \Longrightarrow \phi, \psi_2, \Delta}{\Gamma \Longrightarrow \backslash\text{if } \phi \ \backslash\text{then } \psi_1 \ \backslash\text{else } \psi_2, \Delta}$$

  The rule in KeY is really

$$\frac{\Gamma[\psi_1], \phi \Longrightarrow \Delta[\psi_1] \quad \Gamma[\psi_2] \Longrightarrow \phi, \Delta[\psi_2]}{\Gamma[\backslash\text{if } \phi \ \backslash\text{then } \psi_1 \ \backslash\text{else } \psi_2] \Longrightarrow \Delta[\backslash\text{if } \phi \ \backslash\text{then } \psi_1 \ \backslash\text{else } \psi_2],}$$

  i. e., the if-then-else can be replaced in arbitrary sub-formulas.

Demo

Which formula is equivalent to

- $j = 3 \land k = 5 \rightarrow \langle i = j + k; \text{if } (i < j)\ k = i;\ \text{else } k = j; \rangle p(i, j, k)$?
  Answer: $j = 3 \land k = 5 \rightarrow p(8, 3, 3)$
- $\langle i = j + k; \text{if } (i < j)\ k = i;\ \text{else } k = j; \rangle p(i, j, k)$?
  Answer: \if $k < 0$ \then $p(j + k, j, j + k)$ \else $p(j + k, j, j)$

Proving Programs with Loops

Given a simple loop:

$$\langle\{\textbf{while}(\mathtt{n} > 0)\,\mathtt{n--};\,\}\rangle\mathtt{n} = 0$$

How can we prove that the loop terminates for all $\mathtt{n} \geq 0$ and that $\mathtt{n} = 0$ holds in the final state?

Method (1): Induction

To prove a property $\phi(x)$ for all $x \geq 0$ we can use induction:

- Show $\phi(0)$.
- Show $\phi(x) \implies \phi(x+1)$ for all $x \geq 0$.

This proves that $\forall x\ (x \geq 0 \rightarrow \phi(x))$ holds.

The rule int_induction

The KeY-System has the rule int_induction

$$\frac{\Gamma \Longrightarrow \Delta, \phi(0) \quad \Gamma \Longrightarrow \Delta, \forall X (X \geq 0 \land \phi(X) \to \phi(X+1)) \quad \Gamma, \forall X (X \geq 0 \to \phi(X)) \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta}$$

The three goals are:

- Base Case: $\Longrightarrow \phi(0)$
- Step Case: $\Longrightarrow \forall X (X \geq 0 \land \phi(X) \to \phi(X+1))$
- Use Case: $\forall X (X \geq 0 \to \phi(X)) \Longrightarrow$

Method(2): Loop Invariants with Variants

Induction proofs are very difficult to perform for a loop

$$\langle\{\textbf{while}(COND)\,BODY;\ldots\}\rangle\phi$$

The KeY-system supports special rules for while loops using invariants and variants.

The rule while_invariant_with_variant_dec

The rule while_invariant_with_variant_dec takes an invariant *inv*, a modifies set $\{m_1, \ldots, m_k\}$ and a variant v. The following cases must be proven.

- Initially Valid: $\Longrightarrow inv \land v \geq 0$
- Body Preserves Invariant:

$$\Longrightarrow \{m_1 := x_1 \| \ldots \| m_k := x_k\}(inv \land [\{b = COND;\}]b = \mathbf{true} \\ \rightarrow \langle BODY \rangle inv$$

- Use Case:

$$\Longrightarrow \{m_1 := x_1 \| \ldots \| m_k := x_k\}(inv \land [\{b = COND;\}]b = \mathbf{false} \\ \rightarrow \langle \ldots \rangle \phi$$

- Termination:

$$\Longrightarrow \{m_1 := x_1 \| \ldots \| m_k := x_k\}(inv \land v \geq 0 \land [\{b = COND;\}]b = \mathbf{true} \\ \rightarrow \{old := v\}\langle BODY \rangle v \leq old \land v \geq 0$$