Formal Methods for Java

Lecture 12: Dynamic Logic

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

December 2, 2011

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

Dynamic Logic

Dynamic logic extends predicate logic by

- $[\alpha]\phi$
- $\langle\alpha\rangle\phi$

where α is a program and ϕ a sub-formula.

The meaning is as follows:

- $[\alpha]\phi$: after all terminating runs of program α formula ϕ holds.
- $\langle\alpha\rangle\phi$: after some terminating run of program α formula ϕ holds.

Comparison with Hoare Logic

The sequent $\phi \Longrightarrow [\alpha]\psi$ corresponds to partial correctness of the Hoare formula:

$$\{\phi\}\alpha\{\psi\}$$

If α is deterministic, $\phi \Longrightarrow \langle\alpha\rangle\psi$ corresponds to total correctness.

Examples

- $[\{\}]\phi \equiv \phi$
- $\langle\{\}\rangle\phi \equiv \phi$
- $[\textbf{while}(\textbf{true})\{\}]\phi \equiv \textbf{true}$
- $\langle\textbf{while}(\textbf{true})\{\}\rangle\phi \equiv \textbf{false}$
- $[x = x + 1;]x \geq 4 \equiv x + 1 \geq 4$
- $[x = t;]\phi \equiv \phi[t/x]$
- $[\alpha_1\alpha_2]\phi \equiv [\alpha_1][\alpha_2]\phi$

How can we use equivalences in Sequent Calculus?

Add the rule $\dfrac{\Gamma[\psi/\phi] \Longrightarrow \Delta[\psi/\phi]}{\Gamma \Longrightarrow \Delta}$, where $\phi \equiv \psi$.

This is similar to applyEq.

Dynamic Logic is Modal Logic

- $\langle\alpha\rangle\phi \equiv \neg[\alpha]\neg\phi$
- $[\alpha]\phi \equiv \neg\langle\alpha\rangle\neg\phi$

Furthermore:

- if ϕ is a tautology, so is $[\alpha]\phi$
- $[\alpha](\phi \to \psi) \to ([\alpha]\phi \to [\alpha]\psi)$

Remark: For deterministic programs also the reverse holds

$$([\alpha]\phi \to [\alpha]\psi) \to [\alpha](\phi \to \psi)$$

Termination and Deterministic Programs

How can we express that program α must terminate?

$$\langle\alpha\rangle\textbf{true}$$

This can be used to relate $[\alpha]$ and $\langle\alpha\rangle$:

$$\langle\alpha\rangle\phi \equiv [\alpha]\phi \wedge \langle\alpha\rangle\textbf{true}$$

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* $\rightarrow$ *integer* (mathematical operations)
  - $0, 1, \ldots$: *integer*, *TRUE*, *FALSE* : *boolean* (mathematical constants)
- Non-Rigid Functions: These are functions that depend on current state.
  - $\cdot[\cdot] : \top \times int \rightarrow \top$ (array access)
  - `.next` : $\top \rightarrow \top$ if `next` is a field of a class.
  - `i`, `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.