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 \( \alpha \) is a program and \( \phi \) a sub-formula.

The meaning is as follows:

- \([\alpha] \phi\): after all terminating runs of program \( \alpha \) formula \( \phi \) holds.
- \(\langle \alpha \rangle \phi\): after some terminating run of program \( \alpha \) formula \( \phi \) holds.
Comparison with Hoare Logic

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

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

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

- \([\{}\{}\] \phi \equiv \phi
- \langle \{} \rangle \phi \equiv \phi
- \[\text{while(}\text{true}\{\}\)] \phi \equiv \text{true}
- \langle \text{while(}\text{true}\{\}\)\rangle \phi \equiv \text{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 \(\frac{\Gamma[\psi/\phi] \implies \Delta[\psi/\phi]}{\Gamma \implies \Delta}\), 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 $\phi$ is a tautology, so is $[\alpha] \phi$
- $[\alpha](\phi \rightarrow \psi) \rightarrow ([\alpha] \phi \rightarrow [\alpha] \psi)$

Remark: For deterministic programs also the reverse holds

$$([\alpha] \phi \rightarrow [\alpha] \psi) \rightarrow [\alpha](\phi \rightarrow \psi)$$
How can we express that program $\alpha$ must terminate?

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

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

$$\langle \alpha \rangle \phi \equiv [\alpha] \phi \land \langle \alpha \rangle \text{true}$$
KeY distinguishes the following symbols:

- **Rigid Functions**: These are functions that do not depend on the current state of the program.
  - $+, -, \,* : \text{integer} \times \text{integers} \rightarrow \text{integer}$ (mathematical operations)
  - $0, 1, \ldots : \text{integer}$, $\text{TRUE}$, $\text{FALSE} : \text{boolean}$ (mathematical constants)

- **Non-Rigid Functions**: These are functions that depend on current state.
  - $\cdot[\cdot] : \top \times \text{int} \rightarrow \top$ (array access)
  - $.\text{next} : \top \rightarrow \top$ if $.\text{next}$ 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

\[ \forall x. i = x \rightarrow \langle \{ \text{while}(i > 0)\{i = i - 1; \}\} \rangle i = 0 \]

- 0, 1, - are rigid functions.
- > is a rigid relation.
- i is a non-rigid function.
- x is a logical variable.

Quantification over i is not allowed and x must not appear in a program.
Built-in Rigid Functions

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

Jochen Hoenicke (Software Engineering)
The formula $\langle i = t; \alpha \rangle \phi$ is rewritten to

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

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

Here:

- $i$ is a program variable (non-rigid function).
- $t$ is a term (may contain logical variables).
- $\phi$ a formula
Simplifying Updates

If $\phi$ 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

$$\{x_1 := t'_1[t_1/x_1, t_2/x_2], x_2 := t_2, x_3 := t'_3[t_1/x_1, t_2/x_2]\} \phi$$