The 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
KeY distinguishes the following symbols:

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

- **Non-Rigid Functions**: These are functions that depend on current state.
  - $\cdot \ (\cdot) : \top \times \text{int} \rightarrow \top$ (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.
Builtin 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 $A$. 
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

$$\{x_1 := t'_1[t_1/x_1, t_2/x_2], x_2 := t_2, x_3 := t'_3[t_1/x_1, t_2/x_2]\} \phi$$
Example: $\langle\{i = j; j = i + 1\}\rangle i = j$

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

$$\equiv \{i := j\}\{j := i + 1\} i = j$$

$$\equiv \{i := j, j := j + 1\} i = j$$

$$\equiv j = j + 1$$

$$\equiv false$$

or alternatively

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

$$\equiv \{i := j\}\{j := i + 1\} i = j$$

$$\equiv \{i := j\} i = i + 1$$

$$\equiv j = j + 1$$

$$\equiv false$$
Rules for Java Dynamic Logic

- $\langle\{i = j; \ldots\}\rangle \phi$ is rewritten to:
  $\{i := j\} \langle\{\ldots\}\rangle \phi$.

- $\langle\{i = j + k; \ldots\}\rangle \phi$ is rewritten to:
  $\{i := j + k\} \langle\{\ldots\}\rangle \phi$.

- $\langle\{i = j + +; \ldots\}\rangle \phi$ is rewritten to:
  $\langle\{\textbf{int } j_0; j_0 = j; j = j + 1; i = j_0; \ldots\}\rangle \phi$.

- $\langle\{\textbf{int } k; \ldots\}\rangle \phi$ is rewritten to:
  $\langle\{\ldots\}\rangle \phi$ and $k$ 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:
  \[
  \text{if } i < j \text{ then } \langle\{s_1\}; \ldots\rangle\phi \text{ else } \langle\{s_2\}; \ldots\rangle\phi.
  \]

- $\langle\ldots\text{ if } \ldots \text{ then } \ldots \text{ else } \ldots\rangle$ is a logical operator with the following sequent calculus rules:

  \[
  \begin{align*}
  \Gamma, \phi, \psi_1 & \implies \Delta \quad \Gamma, \psi_2 \implies \phi, \Delta \\
  \Gamma & \implies \langle\ldots\text{ if } \phi \ldots \text{ then } \psi_1 \ldots \text{ else } \psi_2 \ldots\rangle \implies \Delta \\
  \Gamma & \implies \langle\ldots\text{ if } \phi \ldots \text{ then } \psi_1 \ldots \text{ else } \psi_2 \ldots\rangle, \Delta \\
  \Gamma & \implies \langle\ldots\text{ if } \phi \ldots \text{ then } \psi_1 \ldots \text{ else } \psi_2 \ldots\rangle, \Delta \\
  \end{align*}
  \]

  The rule in KeY is really

  \[
  \Gamma[\psi_1], \phi \implies \Delta[\psi_1] \quad \Gamma[\psi_2] \implies \phi, \Delta[\psi_2] \\
  \Gamma[\langle\ldots\text{ if } \phi \ldots \text{ then } \psi_1 \ldots \text{ else } \psi_2 \ldots\rangle] \implies \Delta[\langle\ldots\text{ if } \phi \ldots \text{ then } \psi_1 \ldots \text{ else } \psi_2 \ldots\rangle],
  \]

  i.e., the if-then-else can be replaced in arbitrary sub-formulas.
Which formula is equivalent to

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

- \( \langle i = j + k; \text{if } (i < j) \text{ then } k = i; \text{ else } k = j; \rangle p(i, j, k) \) ?
  
  Answer: \( \text{if } k < 0 \text{ then } p(j + k, j, j + k) \text{ else } p(j + k, j, j) \)
Given a simple loop:

\[
\langle\{\textbf{while}(n > 0)\ n--; \}\rangle n = 0
\]

How can we prove that the loop terminates for all \(n \geq 0\) and that \(n = 0\) holds in the final state?
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`

\[
\begin{align*}
\Gamma \quad &\quad \Delta, \phi(0) & \Gamma \quad &\quad \Delta, \forall X (X \geq 0 \land \phi(X) \rightarrow \phi(X + 1)) \\
\Gamma, \forall X (X \geq 0 \rightarrow \phi(X)) \quad &\quad \Gamma \quad &\quad \Delta
\end{align*}
\]

The three goals are:

- **Base Case:** \( \Rightarrow \phi(0) \)
- **Step Case:** \( \Rightarrow \forall X (X \geq 0 \land \phi(X) \rightarrow \phi(X + 1)) \)
- **Use Case:** \( \forall X (X \geq 0 \rightarrow \phi(X)) \Rightarrow \)

Induction proofs are very difficult to perform for a loop

\[\langle\{\textbf{while}(COND) \textit{BODY}; \ldots}\rangle \phi\]

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:** \( \implies inv \land v \geq 0 \)
- **Body Preserves Invariant:**
  \[
  \implies \{m_1 := x_1 \parallel \ldots \parallel m_k := x_k\}(inv \land [\{b = COND;\}]b = \text{true} \rightarrow \langle BODY\rangle inv
  \]
- **Use Case:**
  \[
  \implies \{m_1 := x_1 \parallel \ldots \parallel m_k := x_k\}(inv \land [\{b = COND;\}]b = \text{false} \rightarrow \langle \ldots\rangle \phi
  \]
- **Termination:**
  \[
  \implies \{m_1 := x_1 \parallel \ldots \parallel m_k := x_k\}(inv \land v \geq 0 \land [\{b = COND;\}]b = \text{true} \rightarrow \{old := v\}\langle BODY\rangle v \leq old \land v \geq 0
  \]