Specification & Formal Analysis of Java Programs
Functional Verification of Java Programs

Prof. Dr. Bernhard Beckert | ADAPT 2010
Dynamic Logic Formulas (Simple Version)

**Definition (Dynamic Logic Formulas (DL Formulas))**

- Each FOL formula is a DL formula
- If $p$ is a program and $\phi$ a DL formula then $\{\langle p \rangle \phi, [p] \phi\}$ 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 (Simple Version)

<table>
<thead>
<tr>
<th>Definition (Dynamic Logic Formulas (DL Formulas))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each FOL formula is a DL formula</td>
</tr>
<tr>
<td>If ( p ) is a program and ( \phi ) a DL formula then ( { \langle p \rangle \phi } ) is a DL formula</td>
</tr>
<tr>
<td>DL formulas closed under FOL quantifiers and connectives</td>
</tr>
</tbody>
</table>

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

- \( \forall \text{int } y; ((\langle x = 1; \rangle x \cdot y) \leftrightarrow (\langle x = 1 \cdot 1; \rangle x \cdot y)) \)
  
  Well-formed if FSym\(_{nr}\) contains int \( x \);

- \( \exists \text{int } x; [x = 1;](x \div 1) \)
  
  Not well-formed, because logical variable occurs in program.

- \( \langle x = 1; \rangle([\text{while (true) {}};] \text{false}) \)
  
  Well-formed if FSym\(_{nr}\) contains int \( x \); program formulas can be nested.
Example (Well-formed? If yes, under which signature?)

- \( \forall \text{int } y; ((\langle x = 1; \rangle x \div y) \iff (\langle x = 1 \ast 1; \rangle x \div y)) \)
  
  Well-formed if FSym\(_{nr}\) contains \text{int } x;

- \( \exists \text{int } x; [x = 1;](x \div 1) \)
  
  Not well-formed, because logical variable occurs in program

- \( \langle x = 1;\rangle([\text{while (true) \{} {};]false) \)
  
  Well-formed if FSym\(_{nr}\) contains \text{int } x;
  
  program formulas can be nested
### Example (Well-formed? If yes, under which signature?)

<table>
<thead>
<tr>
<th>Formula</th>
<th>Well-formed? Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall \text{int } y; (\langle x = 1; \rangle x \doteq y) \leftrightarrow (\langle x = 1 \times 1; \rangle x \doteq y)$</td>
<td>Well-formed if $\text{FSym}_{nr}$ contains $\text{int } x$;</td>
</tr>
<tr>
<td>$\exists \text{int } x; [x = 1;](x \doteq 1)$</td>
<td>Not well-formed, because logical variable occurs in program</td>
</tr>
<tr>
<td>$\langle x = 1; \rangle([\text{while(true)} {};{};\text{false})$</td>
<td>Well-formed if $\text{FSym}_{nr}$ contains $\text{int } x$; program formulas can be nested</td>
</tr>
</tbody>
</table>
Dynamic Logic Formulas Cont’d

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

- $\forall \text{int } y; \ (\langle x = 1; \rangle x \div y) \iff (\langle x = 1 * 1; \rangle x \div y)$
  Well-formed if $\text{FSym}_{nr}$ contains $\text{int } x$;

- $\exists \text{int } x; \ [x = 1;] (x \div 1)$
  Not well-formed, because logical variable occurs in program

- $\langle x = 1; \rangle ([\text{while (true) } \{\};] \text{false})$
  Well-formed if $\text{FSym}_{nr}$ contains $\text{int } x$;
  program formulas can be nested
Example (Well-formed? If yes, under which signature?)

- \( \forall \text{int } y; ((\langle x = 1; \rangle x \doteq y) \leftrightarrow (\langle x = 1 \ast 1; \rangle x \doteq y)) \)
  - Well-formed if \( \text{FSym}_{nr} \) contains \text{int } x;

- \( \exists \text{int } x; [x = 1;](x \doteq 1) \)
  - Not well-formed, because logical variable occurs in program

- \( \langle x = 1; \rangle([\text{while (true) {}};] \text{false}) \)
  - Well-formed if \( \text{FSym}_{nr} \) contains \text{int } x;
  - program formulas can be nested
Example (Well-formed? If yes, under which signature?)

- $\forall \text{int } y; ((\langle x = 1; \rangle x \doteq y) \leftrightarrow (\langle x = 1*1; \rangle x \doteq y))$
  
  Well-formed if $\text{FSym} \_ nr$ contains $\text{int } x$;

- $\exists \text{int } x; [x = 1;](x \doteq 1)$
  
  Not well-formed, because logical variable occurs in program

- $\langle x = 1;\rangle([\text{while } (\text{true}) \{ \};]\text{false})$
  
  Well-formed if $\text{FSym} \_ nr$ contains $\text{int } x$;

program formulas can be nested
Semantic Evaluation of Program Formulas

Definition (Validity Relation for Program Formulas)

- \( s, \beta \models \langle p \rangle \phi \) iff \( \rho(p)(s), \beta \models \phi \) and \( \rho(p)(s) \) is defined
  - \( p \) terminates and \( \phi \) is true in the final state after execution

- \( s, \beta \models [p] \phi \) iff \( \rho(p)(s), \beta \models \phi \) whenever \( \rho(p)(s) \) is defined

If \( p \) terminates then \( \phi \) is true in the final state after execution
Program Correctness

Definition (Notions of Correctness)

- If $s, \beta \models \langle p \rangle \phi$ then
  
  $p$ totally correct (with respect to $\phi$) in $s, \beta$

- If $s, \beta \models [p] \phi$ then
  
  $p$ partially correct (with respect to $\phi$) in $s, \beta$

- Duality $\langle p \rangle \phi$ iff $![p] ! \phi$
  
  Exercise: justify this with help of semantic definitions

- Implication if $\langle p \rangle \phi$ then $[p] \phi$
  
  Total correctness implies partial correctness
  
  - converse is false
  
  - holds only for deterministic programs
Semantics of Sequents

\[ \Gamma = \{\phi_1, \ldots, \phi_n\} \text{ and } \Delta = \{\psi_1, \ldots, \psi_m\} \] sets of program formulas

where all logical variables occur bound

Recall: \( s \models (\Gamma \Rightarrow \Delta) \iff s \models (\phi_1 \land \cdots \land \phi_n) \rightarrow (\psi_1 \mid \cdots \mid \psi_m) \)

Define semantics of DL sequents identical to semantics of FOL sequents

Definition (Validity of Sequents over Program Formulas)

A sequent \( \Gamma \Rightarrow \Delta \) over program formulas is \textit{valid} iff

\[ s \models (\Gamma \Rightarrow \Delta) \text{ in all states } s \]

Consequence for program variables

Initial value of program variables implicitly “universally quantified”
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: $s \models (\Gamma \implies \Delta)$ iff $s \models (\phi_1 \& \cdots \& \phi_n) \rightarrow (\psi_1 | \cdots | \psi_m)$

Define semantics of DL sequents identical to semantics of FOL sequents

**Definition (Validity of Sequents over Program Formulas)**

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

$s \models (\Gamma \implies \Delta)$ in *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 $\text{Init}$ with
   $$ s \models \text{Init} \iff s \in S_0 $$

2. Use sequent
   $$ \Gamma, \text{Init} \Rightarrow \Delta $$
Operational Semantics of Programs

In labelled transition system $K = (S, \rho)$:

$\rho : \Pi \to (S \to S)$ is operationally semantics of programs $p \in \Pi$

How is $\rho$ defined for concrete programs and states?

Example (Operational semantics of assignment)

States $s$ interpret non-rigid symbols $f$ with $I_s(f)$

$\rho(x=t)(s) = s'$ where $s'$ identical to $s$ except $I_{s'}(x) = \text{val}_s(t)$

Very tedious task to define $\rho$ for Java . . .

$\Rightarrow$ go directly to calculus for program formulas!
Operational Semantics of Programs

In labelled transition system $K = (S, \rho)$:

$\rho : \Pi \rightarrow (S \rightarrow S)$ is operational semantics of programs $p \in \Pi$

How is $\rho$ defined for concrete programs and states?

Example (Operational semantics of assignment)

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

$\rho(x = t)(s) = s'$ where $s'$ identical to $s$ except $\mathcal{I}_{s'}(x) = \text{val}_s(t)$

Very tedious task to define $\rho$ for Java . . .

$\Rightarrow$ go directly to calculus for program formulas!
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:

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

\[
\langle \text{stmt}; \text{rest} \rangle \phi, \quad [\text{stmt}; \text{rest}] \phi
\]

- Rules must \textit{symbolically execute} first statement
- Repeated application of rules in a proof corresponds to \textit{symbolic program execution}
**Symbolic Execution of Programs Cont’d**

**Symbolic execution of assignment**

\[
\begin{align*}
\text{assign} \quad & \{x/x_{\text{old}}\} \Gamma, \quad x \doteq \{x/x_{\text{old}}\} t \quad \Rightarrow \quad \langle \text{rest} \rangle \phi, \quad \{x/x_{\text{old}}\} \Delta \\
\Gamma \quad & \Rightarrow \quad \langle x = t; \; \text{rest} \rangle \phi, \Delta
\end{align*}
\]

\( x_{\text{old}} \) new program variable that “rescues” old value of \( x \)

**Example**

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

\[
\begin{align*}
\{x_{\text{old}} \doteq x_0, \; y \doteq y_0, \; x \doteq x_{\text{old}}+y \} \quad & \Rightarrow \quad \langle y=x-y; \; x=x-y;\rangle(x \doteq y_0 \& y \doteq x_0) \\
\{x \doteq x_0, \; y \doteq y_0 \} \quad & \Rightarrow \quad \langle x=x+y; \; y=x-y; \; x=x-y;\rangle(x \doteq y_0 \& y \doteq x_0)
\end{align*}
\]
Symbolic Execution of Programs
Cont’d

Symbolic execution of assignment

\[
\text{assign } \Gamma, x \overset{x/\text{x}_{\text{old}}} \rightarrow t \quad \Rightarrow \quad \langle \text{rest} \rangle \phi, \{x/\text{x}_{\text{old}}\} \Delta
\]

\[
\Gamma \Rightarrow \langle x = t; \text{rest} \rangle \phi, \Delta
\]

\(x_{\text{old}}\) new program variable that “rescues” old value of \(x\)

Example

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

\[
\begin{align*}
\overset{\cdot}{x}_{\text{old}} &= x_0, \ y \overset{\cdot}{=} y_0, \ x \overset{\cdot}{=} x_{\text{old}} + y \Rightarrow \langle y=x-y; \ x=x-y; \phi/(x \overset{\cdot}{=} y_0 \ \& \ y \overset{\cdot}{=} x_0)\rangle \\
\overset{\cdot}{x} &= x_0, \ y \overset{\cdot}{=} y_0 \Rightarrow \langle x=x+y; \ y=x-y; \ x=x-y; \phi/(x \overset{\cdot}{=} y_0 \ \& \ y \overset{\cdot}{=} x_0)\rangle
\end{align*}
\]
Partial correctness assertion

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

In Hoare logic: $\{\text{Pre}\} \ p \ \{\text{Post}\}$ (Pre, Post must be FOL)

In DL: $\text{Pre} \rightarrow [p]\text{Post}$ (Pre, Post any DL formula)

Example (In KeY Syntax, Demo automatic proof)

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

\problem {
    (\forall int x0; \forall int y0; ((x=x0 & y=y0) \rightarrow
    \langle x=x+y; y=x-y; x=x-y; \rangle (x=y0 & y=x0)))
}
```
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 $\{\text{Pre}\} p \{\text{Post}\}$ (Pre, Post must be FOL)

In DL $\text{Pre} \rightarrow [p]\text{Post}$ (Pre, Post any DL formula)

Example (In KeY Syntax, Demo automatic proof)

\begin{verbatim}
\programVariables { 
    int x; int y; }

\problem { 
    (\forall \text{int } x0; \forall \text{int } y0; ((x=x0 & y=y0) -> \langle x=x+y; y=x-y; x=x-y; \rangle (x=y0 & y=x0)))
}
\end{verbatim}
More Properties

Example

\( \forall T \ y; \ ( ((\langle p \rangle x \models y) \iff (\langle q \rangle x \models y)) \)

Not valid in general

Programs \( p \) behave \( q \) equivalently on variable \( T \ x \)

Example

\( \exists T \ y; \ (x \models y \rightarrow \langle p \rangle \text{true}) \)

Not valid in general

Program \( p \) terminates in all states where \( x \) has suitable initial value
More Properties

Example

\[ \forall \; T \; y; \; ((\langle p \rangle x \triangleright y) \iff (\langle q \rangle x \triangleright y)) \]

Not valid in general

Programs \( p \) behave \( q \) equivalently on variable \( T \; x \)

---

Example

\[ \exists \; T \; y; \; (x \triangleright y \rightarrow \langle p \rangle \text{true}) \]

Not valid in general

Program \( p \) terminates in all states where \( x \) has suitable initial value
More Properties

Example

∀ T y; ((⟨p⟩x ⊨ y) ↔ (⟨q⟩x ⊨ y))
Not valid in general
Programs p behave q equivalently on variable T x

Example

∃ T y; (x ⊨ y → ⟨p⟩true)
Not valid in general
Program p terminates in all states where x has suitable initial value
More Properties

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall T \ y; (((\langle p \rangle x \doteq y) \iff (\langle q \rangle x \doteq y)) )</td>
</tr>
<tr>
<td>Not valid in general</td>
</tr>
<tr>
<td>Programs ( p ) behave ( q ) equivalently on variable ( T \ x )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \exists T \ y; (x \doteq y \rightarrow (\langle p \rangle \text{true}) )</td>
</tr>
<tr>
<td>Not valid in general</td>
</tr>
<tr>
<td>Program ( p ) terminates in all states where ( x ) has suitable initial value</td>
</tr>
</tbody>
</table>
Symbolic Execution of Programs
Cont’d

Symbolic execution of conditional

\[ \Gamma, b \models \text{true} \Rightarrow \langle p; \text{rest} \rangle \phi, \Delta \]
\[ \Gamma, b \models \text{false} \Rightarrow \langle q; \text{rest} \rangle \phi, \Delta \]

Symbolic execution must consider all possible execution branches

Symbolic execution of loops: unwind

unwindLoop

\[ \Gamma \Rightarrow \langle \text{if} (b) \{ \ p \ \} \ \text{else} \{ \ q \ \} ; \text{rest} \rangle \phi, \Delta \]
Symbolic Execution of Programs

Cont’d

Symbolic execution of conditional

\[
\Gamma, b \models \text{true} \Rightarrow \langle p; \ \text{rest}\rangle \phi, \Delta \quad \Gamma, b \models \text{false} \Rightarrow \langle q; \ \text{rest}\rangle \phi, \Delta
\]

\[
\Gamma \Rightarrow \langle \text{if} (b) \{ p \} \ \text{else} \{ q \}; \ \text{rest}\rangle \phi, \Delta
\]

Symbolic execution must consider all possible execution branches

Symbolic execution of loops: unwind

\[
\text{unwindLoop} \quad \Gamma \Rightarrow \langle \text{if} (b) \{ p; \ \text{while} (b) p\}; \ r\rangle \phi, \Delta
\]

\[
\Gamma \Rightarrow \langle \text{while} (b) \{ p\}; \ r\rangle \phi, \Delta
\]
How to express correctness for any initial value of program variable?

Not allowed: \( \forall T \ i; \langle \ldots i \ldots \rangle \phi \)  
(program \( \neq \) logical variable)

Not intended: \( \Rightarrow \langle \ldots i \ldots \rangle \phi \)  
(Validity of sequents: quantification over all states)

As previous: \( \forall T \ i_0; \ (i_0 \models i \ \Rightarrow \ \langle \ldots i \ldots \rangle \phi) \)

Solution

Use explicit construct to record values in current state

**Update** \( \forall T \ i_0; \ (\{i := i_0\}\langle \ldots i \ldots \rangle \phi) \)
Quantifying over Program Variables

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

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

Not intended: $\Rightarrow \langle \ldots i \ldots \rangle \phi$ (Validity of sequents: quantification over all states)

As previous: $\forall T \ i_0; \ (i_0 \triangleright i \rightarrow \langle \ldots i \ldots \rangle \phi)$

Solution

Use explicit construct to record values in current state

$Update \quad \forall T \ i_0; \ (\{i := i_0\} \langle \ldots i \ldots \rangle \phi)$
Quantifying over Program Variables

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

Not allowed: \[ \forall T \; i; \langle \ldots i \ldots \rangle \phi \] (program \( \neq \) logical variable)

Not intended: \[ \Rightarrow \langle \ldots i \ldots \rangle \phi \] (Validity of sequents: quantification over \textit{all} states)

As previous: \[ \forall T \; i_0; (i_0 \dot{=} i \rightarrow \langle \ldots i \ldots \rangle \phi) \]

Solution

Use explicit construct to record values in \textit{current} state

\textbf{Update} \[ \forall T \; i_0; (\{i := i_0\}\langle \ldots i \ldots \rangle \phi) \]
Quantifying over Program Variables

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

Not allowed: \( \forall T_{i}; \langle \ldots i \ldots \rangle \phi \)  
(program \( \neq \) logical variable)

Not intended: \( \Rightarrow \langle \ldots i \ldots \rangle \phi \) (Validity of sequents: quantification over all states)

As previous: \( \forall T_{i_{0}}; (i_{0} \triangleq i \rightarrow \langle \ldots i \ldots \rangle \phi) \)

Solution

Use explicit construct to record values in current state

Update \( \forall T_{i_{0}}; \{i := i_{0}\} \langle \ldots i \ldots \rangle \phi \)
How to express correctness for any initial value of program variable?

Not allowed: \( \forall T \ i; \langle \ldots \ i \ldots \rangle \phi \) (program \( \neq \) logical variable)

Not intended: \( \Rightarrow \langle \ldots \ i \ldots \rangle \phi \) (Validity of sequents: quantification over all states)

As previous: \( \forall T \ i_0; (i_0 \vdash i \rightarrow \langle \ldots \ i \ldots \rangle \phi) \)

Solution

Use explicit construct to record values in current state

Update \( \forall T \ i_0; (\{i := i_0\}\langle \ldots \ i \ldots \rangle \phi) \)
Explicit State Updates

Updates specify computation state where formula is evaluated

Definition (Syntax of Updates)
If \( v \) is program variable, \( t \) FOL term type-compatible with \( v \), \( t' \) any FOL term, and \( \phi \) any DL formula, then

- \( \{v := t\}t' \) is DL term
- \( \{v := t\}\phi \) is DL formula

Definition (Semantics of Updates)
State \( s \) interprets non-rigid symbols \( f \) with \( \mathcal{I}_s(f) \)
\( \beta \) variable assignment for logical variables in \( t \)

\[ \rho(\{v := t\})(s) = s' \] where \( s' \) identical to \( s \) except
\[ \mathcal{I}_{s'}(x) = \text{val}_{s,\beta}(t) \]
Explicit State Updates Cont’d

Facts about updates \( \{ v := t \} \)

- 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
  \( \{ x := n \} \phi \) cannot be turned into assignment (\( n \) logical variable)
  \( \langle x=i++; \phi \rangle \) 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

\[
\text{assign} \quad \frac{\Gamma \Rightarrow \{x := t\} \langle \text{rest} \rangle \phi, \Delta}{\Gamma \Rightarrow \langle x = t; \text{rest} \rangle \phi, \Delta}
\]

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

Demo

\text{swap.key}
Example Proof

Example

\texttt{programVariables \{} \texttt{int \ x;} \texttt{\}}

\texttt{problem \{}
\texttt{\exists \texttt{int \ y;}}
\texttt{\{x := y\} \ast \texttt{while \ (x > 0) \{x = x-1;\}\} \rightarrow x=0 \}}

Intuitive Meaning? Satisfiable? Valid?

Demo

term.key

What to do when we \textit{cannot} determine a concrete loop bound?
Example

\texttt{\textbf{Example}}

\begin{verbatim}
\texttt{\textbf{programVariables} { }
  \texttt{int x;}
}
\texttt{\textbf{problem} { }
  \texttt{(exists int y; }
    \texttt{(x := y)\{while (x > 0) {x = x-1;}\} x=0 ))}
\end{verbatim}

Intuitive Meaning? Satisfiable? Valid?

Demo

term.key

What to do when we \textit{cannot} determine a concrete loop bound?
Parallel Updates

How to apply updates on updates?

Example

Symbolic execution of

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

yields:

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

Need to compose three sequential state changes into a single one!
**Definition (Parallel Update)**

A *parallel update* is expression of the form

\[
\{ l_1 := v_1 \parallel \cdots \parallel l_n := v_n \}
\]

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)**

\[
\{ l_1 := r_1 \}\{ l_2 := r_2 \} = \{ l_1 := r_1 \parallel l_2 := \{ l_1 := r_1 \} r_2 \}
\]

\[
\{ l_1 := v_1 \parallel \cdots \parallel l_n := v_n \} x = \left\{ \begin{array}{ll}
  x & \text{if } x \not\in \{ l_1, \ldots, l_n \} \\
  v_k & \text{if } x = l_k, x \not\in \{ l_{k+1}, \ldots, l_n \}
\end{array} \right.
\]
Parallel Updates Cont’d

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)

\[ \{ l_1 := r_1 \} \{ l_2 := r_2 \} = \{ l_1 := r_1 \| l_2 := \{ l_1 := r_1 \} r_2 \} \]
\[ \{ l_1 := v_1 \| \cdots \| l_n := v_n \} x = \begin{cases} x & \text{if } x \notin \{ l_1, \ldots, l_n \} \\ v_k & \text{if } x = l_k, x \notin \{ l_{k+1}, \ldots, l_n \} \end{cases} \]
Parallel Updates Cont’d

Example

\[
\begin{align*}
&\{ \{ x := x+y \} \{ y := x-y \} \} \{ x := x-y \} = \\
&\{ x := x+y \ || \ y := (x+y) - y \} \{ x := x-y \} = \\
&\{ x := x+y \ || \ y := (x+y) - y \ || \ x := (x+y) - ((x+y) - y) \} = \\
&\{ x := x+y \ || \ y := x \ || \ x := y \} = \\
&\{ y := x \ || \ x := y \}
\end{align*}
\]

KeY automatically deletes overwritten (unnecessary) updates

Demo

swap.key

Parallel updates to store intermediate state of symbolic computation
Parallel Updates Cont’d

Example

\[
\begin{align*}
\{x := x+y \downarrow \downarrow y := x-y\} \{x := x-y\} &= \\
\{x := x+y \downarrow y := (x+y)-y\} \{x := x-y\} &= \\
\{x := x+y \downarrow y := (x+y)-y \downarrow x := (x+y)-(x+y)-y\} &= \\
\{x := x+y \downarrow y := x \downarrow x := y\} &= \\
\{y := x \downarrow x := y\} &= 
\end{align*}
\]

KeY automatically deletes overwritten (unnecessary) updates

Demo

\texttt{swap.key}

Parallel updates to store intermediate state of symbolic computation
A Warning

First-order rules that substitute arbitrary terms

\[ \exists \text{-right} \quad \frac{\Gamma \Rightarrow [x/t'] \phi, \exists T x; \phi, \Delta}{\Gamma \Rightarrow \exists T x; \phi, \Delta} \]

\[ \forall \text{-left} \quad \frac{\Gamma, \forall T x; \phi, [x/t'] \phi \Rightarrow \Delta}{\Gamma, \forall T x; \phi \Rightarrow \Delta} \]

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

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

Example

\[ \Gamma, i \doteq 0 \quad \Rightarrow \quad \langle i++ \rangle i \doteq 0 \Rightarrow \Delta \]

\[ \Gamma, \forall T x; (x \doteq 0 \quad \Rightarrow \quad \langle i++ \rangle x \doteq 0) \Rightarrow \Delta \]

Logically valid formula would result in unsatisfiable antecedent!

**Key prohibits unsound substitutions**