Formal Specification and Verification
Reasoning about Programs with Dynamic Logic

Bernhard Beckert

Based on a lecture by Wolfgang Ahrendt and Reiner Hähnle at Chalmers University, Göteborg
Beyond Propositional Logic

Propositional Logic

Temporal Logic

First-order Logic

Dynamic Logic

+computations

+computations +functions

+functions
Beyond Propositional Logic

Propositional Logic

Temporal Logic
  +computations +functions

First-order Logic
  +functions

Dynamic Logic

SPIN

KeY

Today’s lecture

Formal Specification and Verification: Simple DL
B. Beckert 2 / 53
State Dependence of Formula Evaluation
Closed FOL formula is either valid or not valid wrt model $\mathcal{M}$
Consider $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ as program state

Let $x$ be (local) program variable or attribute
Execution of program $p$ may change program state, i.e., value of $x$
State Dependence of Formula Evaluation

Closed FOL formula is either valid or not valid wrt model $M$

Consider $M = (D, \delta, I)$ as program state

Let $x$ be (local) program variable or attribute

Execution of program $p$ may change program state, i.e., value of $x$

Example

Executing $x=3$; results in $M$ such that $M \models x \doteq 3$

Executing $x=4$; results in $M$ such that $M \not\models x \doteq 3$
State Dependence of Formula Evaluation

Closed FOL formula is either valid or not valid with respect to model $\mathcal{M}$

Consider $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ as program state

Let $x$ be (local) program variable or attribute

Execution of program $p$ may change program state, i.e., value of $x$

**Example**

Executing $x=3$; results in $\mathcal{M}$ such that $\mathcal{M} \models x = 3$

Executing $x=4$; results in $\mathcal{M}$ such that $\mathcal{M} \not\models x = 3$

Need a logic to capture state before/after program execution
Rigid versus Flexible Symbols

Signature of program logic defined as in FOL, but:
In addition there are program variables, attributes, etc.

### Rigid versus Flexible

- **Rigid** symbols, same interpretation in all program states
  - First-order variables (aka logical variables)
    Used to hold initial values of program variables
  - Built-in functions and predicates such as $0, 1, \ldots, +, *, \ldots, <, \ldots$

- **Non-rigid** (or flexible) symbols, interpretation depends on state
  Capture side effects on state during program execution
  - Functions modeling program variables and attributes are flexible

Any term containing at least one flexible symbol is also flexible
Signature of Dynamic Logic (Simple Version)

Definition (Dynamic Logic Signature)

First-order signature \( \Sigma = (\text{PSym}_r, \text{FSym}_r, \text{FSym}_{nr}, \alpha) \)

Rigid Predicate Symbols \( \text{PSym} = \{>, \geq, \ldots\} \)

Rigid Function Symbols \( \text{FSym} = \{+, -, *, 0, 1, \ldots, \text{true}, \text{false}\} \)

Non-rigid Function Symbols \( \text{FSym} = \{i, j, k, \ldots, p, q, r, \ldots\} \)

Type hierarchy

\( T = \{\bot, \text{int}, \text{boolean}, \top\} \) with int, boolean incomparable

Standard typing: boolean true; <(int, int); , etc.
Variables

**Definition (First-Order/Logical Variables)**

Typed **logical variables** (rigid), declared as $T \ x$;

**Program Variables**

Non-rigid constants `int i; boolean p` used as **program variables**
Terms

- First-order terms defined as in FOL
- First-order terms may contain rigid and non-rigid symbols
- $\text{FSym}_r \cap \text{FSym}_{nr} = \emptyset$

Example

Signature for $\text{FSym}_{nr}$: int $j$; boolean $p$
Variables int $x$; boolean $b$;

- $j$ and $j + x$ are flexible terms of type int
- $p$ is a flexible term of type boolean
- $x + x$ is a rigid term of type int
- $j + b$ and $j + p$ are not well-typed
Atomic Programs

Definition (Atomic Programs)

The atomic programs $\Pi_0$ are assignments of the form $j = t$ where:

- $T_j$; is a program variable (flexible constant)
- $t$ is a first-order term of type $T$ without logical variables

Example

Signature for $FSym_{nr}$: int $j$; boolean $p$

Variables int $x$; boolean $b$;

- $j = j + 1$, $j = 0$ and $p = \text{false}$ are assignments
- $j = j + x$ contains a logical variable on the right
- $x = 1$ contains a logical variable on the left
- $j = j$ is equality, not assignment
- $p = 0$ is ill-typed
Definition (Program)

Inductive definition of the set of (DL) programs $\Pi$:

- If $\pi$ is an atomic program, then $\pi;$ is a program.
- If $p$ and $q$ are programs, then $pq$ is a program.
- If $b$ is a variable-free term of type boolean, $p$ and $q$ programs, then $\text{if} \ (b) \ p \ \text{else} \ q;$ and $\text{if} \ (b) \ p;$ are programs.
- If $b$ is a variable-free term of type boolean, $p$ a program, then $\text{while} \ (b) \ p;$ is a program.
Definition (Program)

Inductive definition of the set of (DL) programs $\Pi$:

- If $\pi$ is an atomic program, then $\pi;\ $ is a program
- If $p$ and $q$ are programs, then $pq$ is a program
- If $b$ is a variable-free term of type boolean, $p$ and $q$ programs, then $\text{if } (b) \ p \ \text{else } q; \quad \text{if } (b) \ p;$
  are programs
- If $b$ is a variable-free term of type boolean, $p$ a program, then $\text{while } (b) \ p;$
  is a program

Programs contain no logical variables!
Example (Admissible Program)

Signature for $\text{FSym}_{nr}$: int $r$; int $i$; int $n$;
Signature for $\text{FSym}_r$: int $0$; int $+$ (int,int); int $-$ (int,int);
Signature for $\text{PSym}_r$: $<=$ (int,int);

```
i=0;
r=0;
while (i<n) {
    i=i+1;
    r=r+i;
}
```

```
r=r+r-n;
```
Example (Admissible Program)

Signature for $\text{FSym}_{nr}$: \( \text{int } r; \text{ int } i; \text{ int } n; \)
Signature for $\text{FSym}_{r}$: \( \text{int } 0; \text{ int } +_{(\text{int},\text{int})}; \text{ int } -_{(\text{int},\text{int})}; \)
Signature for $\text{PSym}_{r}$: \( <_{(\text{int},\text{int})}; \)

\[
\begin{align*}
i &= 0; \\
r &= 0; \\
\text{while } (i < n) \{ \\
&\quad i = i + 1; \\
&\quad r = r + i; \\
&\} \\
r &= r + r - n;
\end{align*}
\]

Which value does the program compute in \( r \)?
Definition (Dynamic Logic Formulas (DL Formulas))

- Each FOL formula is a DL formula
- If $p$ is a program and $\phi$ a DL formula then $\{\langle p \rangle \phi \}$ is a DL formula
- DL formulas closed under FOL quantifiers and connectives
Definition (Dynamic Logic Formulas (DL Formulas))

- Each FOL formula is a DL formula
- If $p$ is a program and $\phi$ a DL formula then $\{\langle p \rangle \phi \}$ 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
Example (Well-formed? If yes, under which signature?)

\[ \forall \text{int } y; (((x = 1; x \div y) \iff (x = 1*1; x \div y))) \]
Example (Well-formed? If yes, under which signature?)

\[ \forall \text{int } y; (\langle x = 1; \rangle x \div y) \leftrightarrow (\langle x = 1*1; \rangle x \div y) \]

Well-formed if FSym\textsubscript{nr} contains int \( x \);
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 FSym_{nr} contains int \( x \);

\[ \exists \text{int } x; [x = 1;] (x \div 1) \]
Example (Well-formed? If yes, under which signature?)

- $\forall \text{int } y; ((\langle x = 1; \rangle x \div y) \leftrightarrow (\langle x = 1\ast1; \rangle x \div y))$
  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
Example (Well-formed? If yes, under which signature?)

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

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

- \( \langle x = 1; \rangle([\text{while (true) } \{\};] \text{false}) \)
Example (Well-formed? If yes, under which signature?)

- $\forall \text{int } y; (\langle x = 1; \rangle x \div y) \leftrightarrow (\langle x = 1\times1; \rangle x \div y)$
  Well-formed if $\text{FSym}_{nr}$ 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 $\text{FSym}_{nr}$ contains int $x$;
  program formulas can be nested
Dynamic Logic Semantics: States
First-order model can be considered as program state

- Interpretation of non-rigid symbols can vary from state to state (eg, program variables, attribute values)
- Interpretation of rigid symbols is the same in all states (eg, built-in functions and predicates)

States as first-order models
From now, consider program state as first-order model $\mathcal{M} = (D, \delta, I)$

- Only interpretation $I$ of non-rigid symbols in $\text{FSym}_{nr}$ can change
  $\Rightarrow$ only record values of $f \in \text{FSym}_{nr}$: use $s$ (for state) instead of $\mathcal{M}$
- Set of all states $s$ is $S$
Dynamic Logic Semantics: Kripke Structure

**Definition (Kripke Structure (aka Labelled Transition System))**

Kripke structure or Labelled transition system \( K = (S, \rho) \)

- **State** (=first-order model) \( s = (D, \delta, I) \in S \)
- **Transition relation** \( \rho : \Pi \rightarrow (S \rightarrow S) \)
  - \( \rho \) is the operational semantics of programs \( \Pi \)
  - Each program \( p \in \Pi \) transforms a start state \( s \) into end state \( \rho(p)(s) \)
  - \( \rho(p)(s) \) can be undefined: \( p \) does not terminate when started in \( s \)
  - Our programs are deterministic (unlike Promela): \( \rho(p) \) is a function (at most one value)
Dynamic Logic Semantics: Kripke Structure Cont’d

Example (Kripke Structure)

Two programs $p$ and $q$
Show $\rho(p)$ and $\rho(q)$, states $S = \{s_1, \ldots, s_6\}$

When $p$ is started in $s_5$ it terminates in $s_4$, etc.

In general, $\Pi$ and $S$ are infinite!
Definition (Validity Relation for Program Formulas)

\[ s, \beta \models \langle p \rangle \phi \iff \rho(p)(s), \beta \models \phi \text{ and } \rho(p)(s) \text{ is defined} \]

- \( p \) terminates and \( \phi \) is true in the final state after execution

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

- If \( p \) terminates then \( \phi \) is true in the final state after execution
Example (Semantic Evaluation of Program Formulas)

Signature FSym_{nr}: boolean a; boolean b;
Notation: I(x) = T iff x appears in lower compartment

Question 1: s_1 \models \langle p \rangle (a \Downarrow \text{true})?
Example (Semantic Evaluation of Program Formulas)
Signature $\text{FSym}_{nr}$: boolean $a$; boolean $b$;
Notation: $\mathcal{I}(x) = T$ iff $x$ appears in lower compartment

Question 2: $s_1 \models \langle q \rangle (a \equiv \text{true})$ ?
Example (Semantic Evaluation of Program Formulas)

Signature $\text{FSym}_{nr}$: boolean $a$; boolean $b$;
Notation: $\mathcal{I}(x) = T$ iff $x$ appears in lower compartment

$$s_1 \xrightarrow{p} s_2 \xrightarrow{q} s_4 \xrightarrow{p} s_5 \xrightarrow{p} s_6$$

Question 3: $s_5 \models \langle q \rangle (a \models \text{true})$?
Example (Semantic Evaluation of Program Formulas)

Signature FSym_{nr}: boolean a; boolean b;

Notation: I(x) = T iff x appears in lower compartment

Question 4: s_5 \models [q](a \equiv true) ?
Program Correctness

Definition (Notions of Correctness)

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

- If \( s, \beta \models [p] \phi \) then
  p \textbf{ 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

Γ = {φ₁, ..., φₙ} and Δ = {ψ₁, ..., ψₘ} sets of program formulas where all logical variables occur bound

Recall: \( s \models (\Gamma \Rightarrow \Delta) \) iff \( s \models (\phi₁ & \cdots & \phiₙ) \rightarrow (ψ₁ | \cdots | ψₘ) \)

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 valid iff

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

Γ = {φ₁, . . . , φₙ} and ∆ = {ψ₁, . . . , ψₘ} sets of program formulas where all logical variables occur bound

Recall: \( s \models (\Gamma \implies \Delta) \) iff \( s \models (\phi₁ \land \cdots \land \phiₙ) \implies (ψ₁ | \cdots | ψₘ) \)

Define semantics of DL sequents identical to semantics of FOL sequents

Definition (Validity of Sequents over Program Formulas)

A sequent Γ ⇒ ∆ over program formulas is valid iff

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

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 $\Gamma, \text{Init} \Rightarrow \Delta$

Later: simple method for specifying initial value of program variables
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?
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
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

```c
int x; int y; x=x+y; y=x-y; x=x-y;
```
General form of rule conclusions in symbolic execution calculus

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

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

Symbolic execution of assignment

\[
\text{assign } \frac{\{x/x_{\text{old}}\} \Gamma, x \equiv \{x/x_{\text{old}}\} t}{\Gamma} \Rightarrow \langle \text{rest} \rangle \phi, \{x/x_{\text{old}}\} \Delta
\]

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

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

\[
\text{assign } \frac{\{x/x_{\text{old}}\} \Gamma, \ x \doteq \{x/x_{\text{old}}\} t \implies \langle \text{rest} \rangle \phi, \ \{x/x_{\text{old}}\} \Delta}{\Gamma \implies \langle x = t; \ rest \rangle \phi, \Delta}
\]

\(x_{\text{old}}\) 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 \doteq y_0 & y \doteq x_0)\}, \{\Gamma/x \doteq x_0, y \doteq y_0\}, \{\Delta/\emptyset\}

\[
\text{assign } \frac{x_{\text{old}} \doteq x_0, y \doteq y_0, x \doteq x_{\text{old}} + y \implies \langle y=x-y; x=x-y; \rangle (x \doteq y_0 & y \doteq x_0)}{x \doteq x_0, y \doteq y_0 \implies \langle x=x+y; y=x-y; x=x+y; \rangle (x \doteq y_0 & y \doteq x_0)}
\]
Partial correctness assertion

If program \( p \) is started in a state satisfying \( \text{Pre} \) and terminates, then its final state satisfies \( \text{Post} \)

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

**In DL** \( \text{Pre} \rightarrow [p]\text{Post} \) (\( \text{Pre}, \text{Post} \) any DL formula)
Proving Partial Correctness

**Partial correctness assertion**

If program \( p \) is started in a state satisfying \( \text{Pre} \) and terminates, then its final state satisfies \( \text{Post} \)

In **Hoare logic**

\[
\{ \text{Pre} \} \ p \ \{ \text{Post} \} \\
(\text{Pre, Post must be FOL})
\]

In **DL**

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

Example (In KeY Syntax, **Demo** automatic proof)

```
program Variables {
   int x; int y; }

problem {
   (\forall int x0; \forall int y0; ((x=x0 & y=y0) ->
      \{x=x+y; y=x-y; x=x-y;\}\>(x=y0 & y=x0)))
```


More Properties

Example

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

∀ T y; (((⟨p⟩x ⊳ y) ←→ (⟨q⟩x ⊳ y))
Not valid in general
Programs p behave q equivalently on variable T x
More Properties

Example

\[ \forall T \; \forall y; ((p \cdot x \equiv y) \leftrightarrow (q \cdot x \equiv y)) \]

Not valid in general

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

Example

\[ \exists T \; \exists y; (x \equiv y \rightarrow (p)\text{true}) \]
More Properties

Example

\[ \forall T \ y; (((\langle p \rangle x \Rightarrow y) \leftrightarrow (\langle q \rangle x \Rightarrow y))) \]
Not valid in general
Programs \( p \) behave \( q \) equivalently on variable \( T \ x \)

Example

\[ \exists T \ y; (x \Rightarrow y \rightarrow \langle p \rangle \text{true}) \]
Not valid in general
Program \( p \) terminates in all states where \( x \) has suitable initial value
Symbolic execution of conditional

\[
\begin{align*}
\Gamma, b \vdash & \text{true} \; \Rightarrow \; \langle p ; \; \text{rest} \rangle \phi, \Delta \\
\Gamma, b \vdash & \text{false} \; \Rightarrow \; \langle q ; \; \text{rest} \rangle \phi, \Delta \\
\Gamma \; \Rightarrow \; \langle \text{if} (b) \; \{ p \} \; \text{else} \; \{ q \} \; ; \; \text{rest} \rangle \phi, \Delta
\end{align*}
\]

Symbolic execution must consider all possible execution branches.
Symbolic Execution of Programs Cont’d

Symbolic execution of conditional

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

Symbolic execution must consider all possible execution branches

Symbolic execution of loops: unwind

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

How to express correctness for any initial value of program variable?
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)
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)
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: \( \forall T \ i_0; \ (i_0 \vdash i \ \Rightarrow \ \langle \ldots i \ldots \rangle \phi) \)
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: \( \forall T \ i_0; (i_0 \vdash i \rightarrow \langle \ldots i \ldots \rangle \phi) \)

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 $I_s(f)$

$\beta$ variable assignment for logical variables in $t$

$\rho(\{v := t\})(s) = s'$ where $s'$ identical to $s$ except $I_{s'}(x) = val_{s,\beta}(t)$
## Explicit State Updates Cont’d

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

- Update semantics identical to assignment
- Value of update depends on logical variables in \( t \): use \( \beta \)
- 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)
  \( \langle x=i++; \rangle \phi \) cannot directly be turned into update
- Updates are **not equations**: change value of non-rigid terms
Computing Effect of Updates (Automatic)

### Rewrite rules for update followed by ...

<table>
<thead>
<tr>
<th>Type</th>
<th>Rewrite Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>program variable</strong></td>
<td>[ { x := t } y \leadsto y ]</td>
</tr>
<tr>
<td></td>
<td>[ { x := t } x \leadsto t ]</td>
</tr>
<tr>
<td><strong>logical variable</strong></td>
<td>[ { x := t } w \leadsto w ]</td>
</tr>
<tr>
<td><strong>complex term</strong></td>
<td>[ { x := t } f(t_1, \ldots, t_n) \leadsto f({ x := t } t_1, \ldots, { x := t } t_n) ]</td>
</tr>
<tr>
<td></td>
<td>[ { x := t } (\phi \land \psi) \leadsto { x := t } \phi \land { x := t } \psi ]</td>
</tr>
<tr>
<td><strong>FOL formula</strong></td>
<td>[ { x := t } (\forall T y; \phi) \leadsto \forall T y; ({ x := t } \phi) ]</td>
</tr>
<tr>
<td><strong>program formula</strong></td>
<td>[ { x := t } (\langle p \rangle \phi) \leadsto { x := t } (\langle p \rangle \phi) ] unchanged!</td>
</tr>
</tbody>
</table>

**Update computation delayed until p symbolically executed**
Assignment Rule Using Updates

Symbolic execution of assignment using updates

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

- 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

Examples/lect11/swap.key
Example Proof

Example

```
\programVariables { 
  int x;
}
\problem { 
  (\exists int y;  
   ({x := y}\{\text{while} (x > 0) \{x = x-1;\}\} } \ x=0 ))
}

Intuitive Meaning? Satisfiable? Valid?
```

Demo

Examples/lect11/term.key
Example

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

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

Intuitive Meaning? Satisfiable? Valid?

Demo

Examples/lect11/term.key

What to do when we cannot determine a concrete loop bound?
Parallel Updates

How to apply updates on updates?

Example
Symbolic execution of

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

yields:

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

Need to compose three sequential state changes into a single one!
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 (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)

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

\[
\{ l_1 := v_1 \parallel \cdots \parallel l_n := v_n \} x = \begin{cases} 
    x & \text{if } x \not\in \{ l_1, \ldots, l_n \} \\
    v_k & \text{if } x = l_k, x \not\in \{ l_{k+1}, \ldots, l_n \}
\end{cases}
\]
Example

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

KeY automatically deletes overwritten (unnecessary) updates

Demo

Examples/lect11/swap.key
Parallel Updates Cont’d

Example

\[\{(x := x+y)(y := x-y)\}(x := x-y) =\]
\[\{x := x+y \parallel y := (x+y)-y\}(x := x-y) =\]
\[\{x := x+y \parallel y := (x+y)-y \parallel x := (x+y)-((x+y)-y)\} =\]
\[\{x := x+y \parallel y := x \parallel x := y\} =\]
\[\{y := x \parallel x := y\}\]

KeY automatically deletes overwritten (unnecessary) updates

Demo

Examples/lect11/swap.key

Parallel updates to store intermediate state of symbolic computation
A Warning

First-order rules that substitute arbitrary terms

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

applyEq \quad \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

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

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

Logically valid formula would result in unsatisfiable antecedent!
KeY prohibits unsound substitutions
Essential

**KeY Book** Verification of Object-Oriented Software (see course web page), Chapter 10: **Using KeY**

**KeY Book** Verification of Object-Oriented Software (see course web page), Chapter 3: **Dynamic Logic** (Sections 3.1, 3.2, 3.4, 3.5, 3.6.1, 3.6.3, 3.6.4)