15-819M: Data, Code, Decisions
07: Reasoning about While Programs with Dynamic Logic

André Platzer

aplatzer@cs.cmu.edu
Carnegie Mellon University, Pittsburgh, PA
Outline

1. DL Syntax
   - State Dependence
   - Signature
   - Terms
   - Atomic Programs
   - DL Programs
   - Program Formulas

2. DL Semantics
   - States
   - Kripke Structures
   - Program Formula Valuation
   - Program Correctness

3. Operational Semantics

4. Symbolic Execution
   - Updates
   - Parallel Updates
   - Restrictions

5. Literature
Outline

1 DL Syntax
   - State Dependence
   - Signature
   - Terms
   - Atomic Programs
   - DL Programs
   - Program Formulas

2 DL Semantics
   - States
   - Kripke Structures
   - Program Formula Valuation
   - Program Correctness

3 Operational Semantics

4 Symbolic Execution
   - Updates
   - Parallel Updates
   - Restrictions

5 Literature
Beyond Propositional Logic

- Propositional Logic
- Temporal Logic
- First-order Logic
- Dynamic Logic

+ state change
+ functions
Beyond Propositional Logic

Propositional Logic

First-order Logic

Dynamic Logic

Temporal Logic

+state change +functions

+functions

Spin

Andre Platzer (CMU)
Beyond Propositional Logic

- Propositional Logic
- Temporal Logic
- First-order Logic
- Dynamic Logic

- +state change
- +functions

Spin

KeY

André Platzer (CMU)

15-819M/07: Data, Code, Decisions
Beyond Propositional Logic

Propositional Logic

First-order Logic

Dynamic Logic

Temporal Logic

KeY

today

SPIN

+state change
+functions
+state change
+functions
+functions
State Dependence of Formula Evaluation

Closed FOL formula either evaluates to true or false in a model $\mathcal{M}$
Consider model $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ 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 either evaluates to true or false in a model $\mathcal{M}$.
Consider model $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ 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 $\mathcal{M} \models x \equiv 3$.
Executing $x = 4$; results in $\mathcal{M}$ such that $\mathcal{M} \not\models x \equiv 3$. 
State Dependence of Formula Evaluation

Closed FOL formula either evaluates to true or false in a model $\mathcal{M}$
Consider model $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ 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 $\mathcal{M} \models x \equiv 3$
Executing $x = 4$; results in $\mathcal{M}$ such that $\mathcal{M} \not\models x \equiv 3$

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.

<table>
<thead>
<tr>
<th>Rigid versus Flexible Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rigid</strong> symbols have same interpretation in all program states</td>
</tr>
<tr>
<td>- First-order variables (logical variables for quantification)</td>
</tr>
<tr>
<td>- Used to hold initial values of program variables</td>
</tr>
<tr>
<td>- Built-in functions and predicates such as $0, 1, \ldots, +, \times, \ldots, &lt;, \ldots$</td>
</tr>
<tr>
<td><strong>Flexible</strong> (or non-rigid) symbols where interpretation depends on state</td>
</tr>
<tr>
<td>- Capture side effects on state during program execution</td>
</tr>
<tr>
<td>- Functions modeling program variables and attributes are flexible</td>
</tr>
</tbody>
</table>

Any term containing at least one flexible symbol is also flexible
Definition (Dynamic Logic Signature)

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

- **Rigid predicate symbols** $\text{PSym} = \{>, \ge, \ldots\}$
- **Rigid function symbols** $\text{FSym} = \{+, -, \ast, 0, 1, \ldots, \text{true}, \text{false}\}$
- **Flexible 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.
Definition (First-Order/Logical Variables)

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

Definition

Program Variables **Flexible** constants int i; boolean p used as program variables
First-order terms defined as in FOL
First-order terms may contain rigid and flexible symbols
\( \text{FSym}_r \cap \text{FSym}_{nr} = \emptyset \)

Example

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

- \( j \) and \( j + x \) are flexible terms of type \( \text{int} \)
- \( p \) is a flexible term of type \( \text{boolean} \)
- \( x + x \) is a rigid term of type \( \text{int} \)
- \( j + b \) 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:

- $\mathbb{T} j$; is a program variable (flexible constant)
- $t$ is a first-order term of type $\mathbb{T}$ without logical variables

Example

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

Variables \text{int} $x$; \text{boolean} $b$;

- $j=j+1$, \hspace{1em} $j=0$ and \hspace{1em} $p=false$ are assignments
- $j=j+x$ contains a logical variable on the right
- $x=1$ contains a logical variable on the left
- $j\div j$ is equality, not assignment
- $p=0$ is ill-typed
Dynamic Logic Programs (Simple Version)

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
Dynamic Logic Programs (Simple Version)

**Definition (Program)**

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

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

Programs contain no logical variables!
Example (Admissible Program)

Signature for $F_{Sym_{nr}}$: int $r$; int $i$; int $n$;
Signature for $F_{Sym_r}$: int 0; int +(int,int); int -(int,int);
Signature for $P_{Sym_r}$: <(int,int);

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

Which value does the program compute in $r$?
Example (Admissible Program)

Signature for FSym_{nr}: \text{int } r; \text{int } i; \text{int } n;
Signature for FSym_{r}: \text{int } 0; \text{int } +({\text{int, int}}); \text{int } -({\text{int, int}});
Signature for PSym_{r}: <({\text{int, int}});

\begin{verbatim}
i=0;
r=0;
while (i<n) {
    i=i+1;
    r=r+i;
};
r=r+r-n;
\end{verbatim}

Which value does the program compute in r?
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 \( \langle p \rangle \phi \) is a DL formula.
- If \( p \) is a program and \( \phi \) a DL formula then \( [p] \phi \) 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
- If $p$ is a program and $\phi$ a DL formula then $[p] \phi$ 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

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

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

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

Well-formed if \( \text{FSym}_{nr} \) contains \text{int } x;
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 $\text{FSym}_{nr}$ contains $\text{int } x$;

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

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

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

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

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

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

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

André Platzer (CMU)
Outline

1. DL Syntax
   - State Dependence
   - Signature
   - Terms
   - Atomic Programs
   - DL Programs
   - Program Formulas

2. DL Semantics
   - States
   - Kripke Structures
   - Program Formula Valuation
   - Program Correctness

3. Operational Semantics

4. Symbolic Execution
   - Updates
   - Parallel Updates
   - Restrictions

5. Literature
Dynamic Logic Semantics: States

First-order model can be considered as program state

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

States as first-order models

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

- Only interpretation $\mathcal{I}$ of flexible symbols in $\text{FSym}_{nr}$ can change
  $\Rightarrow$ only track values of $f \in \text{FSym}_{nr}$: use $s$ (for state) instead of $\mathcal{M}$
- Set of all states $s$ is $S$
Definition (Kripke Structure / 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 (partial) function (at most one value)
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!
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
Example (Semantic Evaluation of Program Formulas)

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

Question 1: \( s_1 \models \langle p \rangle (a \models true) \) ?
Dynamic Logic Semantics: Kripke Structure

Example (Semantic Evaluation of Program Formulas)

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

Question 2: $s_1 \models \langle q \rangle (a \models \text{true})$?
Dynamic Logic Semantics: Kripke Structure

Example (Semantic Evaluation of Program Formulas)

Signature $\text{FSym}_{nr}$: boolean $a$; boolean $b$;

Notation: $\mathcal{I}(x) = T$ iff $x$ appears in node

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

Signature \( \text{FSym}_{nr} \): \text{boolean} \ a; \text{boolean} \ b;

Notation: \( \mathcal{I}(x) = T \) iff \( x \) appears in node

Question 4: \( s_5 \models [q](a \triangleright \text{true}) \) ?
Program Correctness

**Definition (Notions of Program 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 using the semantics

**Implication**

If $\langle p \rangle \phi$ then $[p] \phi$

Total correctness implies partial correctness

- converse is false
- holds only for deterministic programs!
Γ = \{φ_1, \ldots, φ_n\} and Δ = \{ψ_1, \ldots, ψ_m\} sets of program formulas
where all logical variables occur bound

Recall: \(s \models (Γ \Rightarrow Δ)\) iff \(s \models (φ_1 \land \cdots \land φ_n) \Rightarrow (ψ_1 | \cdots | ψ_m)\)

Define semantics of DL sequents like semantics of FOL sequents

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

A sequent \(Γ \Rightarrow Δ\) over program formulas is valid iff

\[ s \models (Γ \Rightarrow Δ) \text{ in all states } s \]
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: \( s \models (\Gamma \Rightarrow \Delta) \) iff \( s \models (\phi_1 \& \cdots \& \phi_n) \rightarrow (\psi_1 \mid \cdots \mid \psi_m) \)

Define semantics of DL sequents like semantics of FOL sequents

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

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

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

**Consequence for program variables**

Initial value of program variables implicitly “universally quantified”
**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 `Init` with
   \[ s \models Init \iff s \in S_0 \]

2. Use sequent \( \Gamma, \text{Init} \Rightarrow \Delta \)

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

1. DL Syntax
   - State Dependence
   - Signature
   - Terms
   - Atomic Programs
   - DL Programs
   - Program Formulas

2. DL Semantics
   - States
   - Kripke Structures
   - Program Formula Valuation
   - Program Correctness

3. Operational Semantics

4. Symbolic Execution
   - Updates
   - Parallel Updates
   - Restrictions

5. Literature

André Platzer (CMU)
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?
Operational Semantics of Programs

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

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

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

Example (Operational semantics of assignment)

State $s$ interprets flexible symbols $f$ with $I_s(f)$

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

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

$\Rightarrow$ here we go directly to calculus for program formulas!
Outline

1 DL Syntax
   - State Dependence
   - Signature
   - Terms
   - Atomic Programs
   - DL Programs
   - Program Formulas

2 DL Semantics
   - States
   - Kripke Structures
   - Program Formula Valuation
   - Program Correctness

3 Operational Semantics

4 Symbolic Execution
   - Updates
   - Parallel Updates
   - Restrictions

5 Literature
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 analyzing 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;
```
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 analyzing 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;
```
Symbolic Execution of Programs

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

Symbolic execution of assignment

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

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

Symbolic execution of assignment

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

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

Example

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

\[
\begin{align*}
x_{\text{old}} & \equiv x_0, \; y \equiv y_0, \; x \equiv x_{\text{old}} + y \quad \Rightarrow \quad \langle y=x-y; \; x=x-y; \rangle (x \equiv y_0 & y \equiv x_0) \\
x \equiv x_0, \; y \equiv y_0 & \Rightarrow \quad \langle x=x+y; \; y=x-y; \; x=x-y; \rangle (x \equiv y_0 & y \equiv x_0)
\end{align*}
\]
### 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**

$$\{\text{Pre}\} \ p \ \{\text{Post}\} \quad (\text{Pre, Post must be FOL})$$

**In DL**

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

$\{\text{Pre}\} \ p \ \{\text{Post}\}$  

(Pre, Post must be FOL)

In DL

$\text{Pre} \rightarrow [p]\text{Post}$  

(Pre, Post any DL formula)

Example (In KeY Syntax, automatic proof)

\begin{verbatim}
\program{} {
  \variables{
    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)))
}

lect11swap.key
\end{verbatim}

(Demo)
Advanced 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 equivalently on variable \( T \)

Example

\[ \exists T \ y; (x = y - \langle p \rangle true) \]

Not valid in general

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

Example

∀ T y; ((⟨p⟩x ⊦ y) ↔ ⟨q⟩x ⊦ y))

Not valid in general

Programs p behave q equivalently on variable T x
Example

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

Not valid in general

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

Example

\[ \exists T \ y; (x \伊利 y \Rightarrow \langle p \rangle \text{true}) \]
Advanced 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

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

Not valid in general

Program \( p \) terminates in all states where \( x \) has suitable initial value
Symbolic execution of conditional

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

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

Symbolic execution must consider all possible execution branches.
Symbolic execution of Programs

Symbolic execution of conditional

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

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

Symbolic execution must consider all possible execution branches

Symbolic execution of loops: unwind

unwindLoop

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

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

Solution
Use explicit construct to record values in current state
Update
\[ \forall T i_0; (\{ i \leftarrow i_0 \} \langle \ldots i \ldots \rangle \phi) \]
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)
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: \( \forall T \ i_0; (i_0 \doteq i \rightarrow \langle \ldots i \ldots \rangle \phi) \)
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: \( \forall T \ i_0; (i_0 \doteq i \rightarrow \langle \ldots i \ldots \rangle \phi) \)

Solution

Use explicit construct to record values in current state

Update \( \forall T \ i_0; (\{ i \doteq i_0 \}\langle \ldots i \ldots \rangle \phi) \)
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 flexible 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

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 flexible terms
Computing Effect of Updates (Automatic)

Rewrite rules for update followed by ...

program variable \( \{ x := t \} y \leadsto y \) where \( x \neq y \)

\( \{ x := t \} x \leadsto t \)

logical variable \( \{ x := t \} w \leadsto w \)

complex term \( \{ x := t \} f(t_1, \ldots, t_n) \leadsto f(\{ x := t \} t_1, \ldots, \{ x := t \} t_n) \)

\( \{ x := t \}(\phi \land \psi) \leadsto \{ x := t \} \phi \land \{ x := t \} \psi \)

\( \{ x := t \}(\forall T y; \phi) \leadsto \forall T y; (\{ x := t \} \phi) \)

FOL formula

program formula \( \{ x := t \}(\langle p \rangle \phi) \leadsto \{ x := t \}(\langle p \rangle \phi) \) unchanged!

Update computation delayed until \( p \) symbolically executed
Assignment Rule Using Updates

Symbolic execution of assignment using updates

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

- Avoids renaming of program variables
- Works as long as \( t \) has no side effects (ok in simple DL)

**Demo**

Examples/lect11/swap.key
Example

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

Intuitive Meaning? Satisfiable? Valid?

Demo

Examples/lect11/term.key
Example Proof

Example

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

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

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

```c
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!
Definition (Parallel Update)

A **parallel update** is an expression of the form \( \{ l_1 := v_1 || \cdots || l_n := v_n \} \) where each \( \{ l_i := v_i \} \) is a simple update:

- All \( v_i \) computed in the old state before the update is applied.
- Updates of all locations \( l_i \) executed simultaneously.
- Upon **conflict** \( l_i = l_j, v_i \neq v_j \) the later update (max \{ \( i, j \) \}) wins.
Parallel Updates

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)

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

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

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

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 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 \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
**Literature for this Lecture**

<table>
<thead>
<tr>
<th>Essential</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KeY Book</strong> Verification of Object-Oriented Software, Chapter 10: <strong>Using KeY</strong></td>
</tr>
<tr>
<td><strong>KeY Book</strong> Verification of Object-Oriented Software, Chapter 3: <strong>Dynamic Logic</strong> (Sections 3.1, 3.2, 3.4, 3.5, 3.6.1, 3.6.3, 3.6.4)</td>
</tr>
</tbody>
</table>