Introduction to Dynamic Logics

Prof. P. H. Schmitt, Christian Engel, Benjamin Weiß

December 3, 2008
Establish $\Gamma \models G$ by purely syntactic transformations of $\Gamma$ and $G$

(Logic) Calculus: a set of transformation rules $\mathcal{R}$ defining relation $\vdash \subseteq 2^{\text{For}_0^\Sigma} \times \text{For}_0^\Sigma$ such that $\Gamma \models G$ iff $\Gamma \vdash G$

$\models \subseteq \vdash$ Completeness $\models \supseteq \vdash$ Soundness

Sequent Calculus based on notion of sequent

\[ \psi_1, \ldots, \psi_m \quad \Rightarrow \quad \phi_1, \ldots, \phi_n \]

Antecedent

Succedent

has same semantics as

\[ (\psi_1 \& \cdots \& \psi_m) \quad \Rightarrow \quad (\phi_1 \mid \cdots \mid \phi_n) \]

\[ \{\psi_1, \ldots, \psi_m\} \models \phi_1 \mid \cdots \mid \phi_n \]
Notation for Sequents

\( \psi_1, \ldots, \psi_m \implies \phi_1, \ldots, \phi_n \)

Consider antecedent/succedent as sets of formulas, may be empty.

Use schematic variables \( \Gamma, \Delta \) that match sets of formulas

\( \Gamma \implies \Delta, \phi \)

Matches any sequent with an occurrence of \( \phi \) in succedent.

Call \( \phi \) main formula and \( \Gamma \) side formulas of sequent.

Any sequent of the form \( \Gamma, \phi \implies \Delta, \phi \) is valid: axiom.
Sequent Calculus Rules

**Basic idea** write syntactic transformation pattern for sequents that mimicks semantics of connectives as closely as possible

**Rule Name**

\[
\begin{align*}
\text{Premisses} & : \Gamma_1 \implies \Delta_1 \quad \cdots \quad \Gamma_r \implies \Delta_r \\
\text{Conclusion} & : \Gamma \implies \Delta
\end{align*}
\]
**Basic idea** write syntactic transformation pattern for sequents that mimicks semantics of connectives as closely as possible

**Rule Name**

\[
\frac{\Gamma_1 \Rightarrow \Delta_1 \quad \cdots \quad \Gamma_r \Rightarrow \Delta_r}{\Gamma \Rightarrow \Delta}
\]

**Premisses**
\[
\{\Gamma_1 \Rightarrow \Delta_1, \cdots, \Gamma_r \Rightarrow \Delta_r\} \models (\Gamma \Rightarrow \Delta)
\]

**Conclusion**
Sequent Calculus Rules

**Basic idea** write syntactic transformation pattern for sequents that mimicks semantics of connectives as closely as possible

**RULE NAME**

\[
\begin{array}{c}
\Gamma_1 \implies \Delta_1 \\
\vdots \\
\Gamma_r \implies \Delta_r \\
\Gamma \implies \Delta
\end{array}
\]

Premisses

Conclusion

Sound rule (essential)
\[
\{ \Gamma_1 \implies \Delta_1, \ldots, \Gamma_r \implies \Delta_r \} \models (\Gamma \implies \Delta)
\]

Complete rule (desirable)
\[
\{ \Gamma \implies \Delta \} \models (\Gamma_1 \implies \Delta_1 \& \ldots \& \Gamma_r \implies \Delta_r)
\]
**Basic idea** write syntactic transformation pattern for sequents that mimicks semantics of connectives as closely as possible

\[
\begin{array}{c}
\text{RULE NAME} \\
\Gamma_1 \implies \Delta_1 \quad \cdots \quad \Gamma_r \implies \Delta_r \\
\hline
\Gamma \implies \Delta
\end{array}
\]

Sound rule (essential)
\[
\{\Gamma_1 \implies \Delta_1, \cdots, \Gamma_r \implies \Delta_r\} \models (\Gamma \implies \Delta)
\]

Complete rule (desirable)
\[
\{\Gamma \implies \Delta\} \models (\Gamma_1 \implies \Delta_1 \land \cdots \land \Gamma_r \implies \Delta_r)
\]

Admissible to have no premisses (iff conclusion is valid, eg axiom)
Sequent Calculus Proofs

Goal to prove: \( S = \psi_1, \ldots, \psi_m \implies \phi_1, \ldots, \phi_n \)

- find rule \( \mathcal{R} \) whose conclusion matches \( S \)

Goal-directed proof search

In KeY tool proof displayed as Java Swing tree
Sequent Calculus Proofs

**Goal** to prove: \( S = \psi_1, \ldots, \psi_m \implies \phi_1, \ldots, \phi_n \)

- find rule \( \mathcal{R} \) whose conclusion matches \( S \)
- instantiate \( \mathcal{R} \) such that conclusion identical to \( S \)
**Sequent Calculus Proofs**

**Goal** to prove: \( S = \psi_1, \ldots, \psi_m \Rightarrow \phi_1, \ldots, \phi_n \)

- find rule \( \mathcal{R} \) whose conclusion matches \( S \)
- instantiate \( \mathcal{R} \) such that conclusion identical to \( S \)
- recursively find proofs for resulting premisses \( S_1, \ldots, S_r \)
Sequent Calculus Proofs

**Goal** to prove: $S = \psi_1, \ldots, \psi_m \Rightarrow \phi_1, \ldots, \phi_n$

- find rule $R$ whose conclusion matches $S$
- instantiate $R$ such that conclusion identical to $S$
- recursively find proofs for resulting premisses $S_1, \ldots, S_r$
- tree structure with goal as root

Goal-directed proof search
In KeY tool proof displayed as Java Swing tree
Sequent Calculus Proofs

**Goal** to prove: \( S = \psi_1, \ldots, \psi_m \Rightarrow \phi_1, \ldots, \phi_n \)

- find rule \( R \) whose conclusion matches \( S \)
- instantiate \( R \) such that conclusion identical to \( S \)
- recursively find proofs for resulting premisses \( S_1, \ldots, S_r \)
- tree structure with goal as root
- **close** proof branch when rule without premise encountered
Sequent Calculus Proofs

**Goal** to prove: \( S = \psi_1, \ldots, \psi_m \implies \phi_1, \ldots, \phi_n \)

- find rule \( R \) whose conclusion matches \( S \)
- instantiate \( R \) such that conclusion identical to \( S \)
- recursively find proofs for resulting premisses \( S_1, \ldots, S_r \)
- tree structure with goal as root
- **close** proof branch when rule without premise encountered

**Goal-directed proof search**
In KeY tool proof displayed as **JAVA Swing tree**
### Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>( \Gamma \implies A, \Delta )</td>
<td>( \Gamma, A \implies \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, !A \implies \Delta )</td>
<td>( \Gamma \implies !A, \Delta )</td>
</tr>
</tbody>
</table>
# Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>( \Gamma \Rightarrow A, \Delta )</td>
<td>( \Gamma, A \Rightarrow \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, \neg A \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \neg A, \Delta )</td>
</tr>
<tr>
<td>and</td>
<td>( \Gamma, A, B \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow A, \Delta ) ( \Gamma \Rightarrow B, \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, A &amp; B \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow A &amp; B, \Delta )</td>
</tr>
</tbody>
</table>
## Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>( \Gamma \Rightarrow A, \Delta )</td>
<td>( \Gamma, A \Rightarrow \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, \neg A \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \neg A, \Delta )</td>
</tr>
<tr>
<td>and</td>
<td>( \Gamma, A, B \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow A, \Delta ) ( \Gamma \Rightarrow B, \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, A &amp; B \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow A &amp; B, \Delta )</td>
</tr>
<tr>
<td>or</td>
<td>( \Gamma, A \Rightarrow \Delta ) ( \Gamma, B \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow A, B, \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, A \mid B \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow A \mid B, \Delta )</td>
</tr>
</tbody>
</table>
### Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>( \Gamma \implies A, \Delta ) ( \Gamma, \neg A \implies \Delta )</td>
<td>( \Gamma, A \implies \Delta ) ( \Gamma \implies \neg A, \Delta )</td>
</tr>
<tr>
<td>and</td>
<td>( \Gamma, A, B \implies \Delta ) ( \Gamma, A \land B \implies \Delta )</td>
<td>( \Gamma \implies A, \Delta ) ( \Gamma \implies B, \Delta ) ( \Gamma \implies A \land B, \Delta )</td>
</tr>
<tr>
<td>or</td>
<td>( \Gamma, A \implies \Delta ) ( \Gamma, B \implies \Delta ) ( \Gamma, A \lor B \implies \Delta )</td>
<td>( \Gamma \implies A, \Delta ) ( \Gamma \implies B, \Delta ) ( \Gamma \implies A \lor B, \Delta )</td>
</tr>
<tr>
<td>imp</td>
<td>( \Gamma \implies A, \Delta ) ( \Gamma, B \implies \Delta ) ( \Gamma, A \implies B, \Delta ) ( \Gamma \implies A \rightarrow B, \Delta )</td>
<td></td>
</tr>
</tbody>
</table>

**Introduction to Dynamic Logics**

December 3, 2008 6 / 37
### Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>not</strong></td>
<td>$\Gamma \Rightarrow A, \Delta$</td>
<td>$\Gamma, A \Rightarrow \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \neg A \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \neg A, \Delta$</td>
</tr>
<tr>
<td><strong>and</strong></td>
<td>$\Gamma, A, B \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow A, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, A \land B \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow B, \Delta$</td>
</tr>
<tr>
<td><strong>or</strong></td>
<td>$\Gamma \Rightarrow A, \Delta$</td>
<td>$\Gamma \Rightarrow A, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, A \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow A \lor B, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, B \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow A \lor B, \Delta$</td>
</tr>
<tr>
<td><strong>imp</strong></td>
<td>$\Gamma \Rightarrow A, \Delta$</td>
<td>$\Gamma \Rightarrow A \Rightarrow B, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, B \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow A \Rightarrow B, \Delta$</td>
</tr>
</tbody>
</table>

**AXIOM**

$\Gamma, A \Rightarrow A, \Delta$

**TRUE**

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

**FALSE**

$\Gamma, \text{false} \Rightarrow \Delta$
Justification of Rules

Compute rules by applying semantics definition of connectives
Compute rules by applying semantics definition of connectives

**OR_RIGHT**

\[
\Gamma \Rightarrow A, B, \Delta \\
\Gamma \Rightarrow A \mid B, \Delta
\]

**AND_RIGHT**

\[
\Gamma \Rightarrow A, \Delta \quad \Gamma \Rightarrow B, \Delta \\
\Gamma \Rightarrow A \& B, \Delta
\]
Justification of Rules

Compute rules by applying semantics definition of connectives

\[
\text{OR\_RIGHT} \quad \Gamma \Rightarrow A, B, \Delta \\
\Gamma \Rightarrow A|B, \Delta
\]

\[
\text{AND\_RIGHT} \quad \Gamma \Rightarrow A, \Delta \quad \Gamma \Rightarrow B, \Delta \\
\Gamma \Rightarrow A\&B, \Delta
\]

Follows directly from semantics of sequents

\[
\Gamma \Rightarrow (A\&B)|\Delta
\]

iff

\[
\Gamma \Rightarrow A|\Delta \quad \text{and} \quad \Gamma \Rightarrow B|\Delta
\]
A Simple Proof

\[ \Gamma \implies (A \& (A \to B)) \to B, \Delta \]
A Simple Proof

\[ \Gamma, (A \& (A \rightarrow B)) \Rightarrow B, \Delta \]
\[ \Gamma \Rightarrow (A \& (A \rightarrow B)) \Rightarrow B, \Delta \]
A Simple Proof

\[
\begin{align*}
\Gamma, A, (A\rightarrow B) & \Rightarrow B, \Delta \\
\Gamma, (A \& (A \rightarrow B)) & \Rightarrow B, \Delta \\
\Gamma & \Rightarrow (A \& (A \rightarrow B)) \rightarrow B, \Delta
\end{align*}
\]
A Simple Proof

\[
\begin{array}{c}
\Gamma, A \implies B, A, \Delta \\
\Gamma, A, (A \implies B) \implies B, \Delta \\
\Gamma, (A \& (A \implies B)) \implies B, \Delta \\
\Gamma \implies (A \& (A \implies B)) \implies B, \Delta
\end{array}
\]
A Simple Proof

\[
\begin{array}{c}
\Gamma, A \implies B, A, \Delta \\
\hline
\Gamma, A, (A \implies B) \implies B, \Delta \\
\hline
\Gamma, (A \& (A \implies B)) \implies B, \Delta \\
\hline
\Gamma \implies (A \& (A \implies B)) \implies B, \Delta \\
\end{array}
\]

A proof is closed, if all its branches are closed.
A Simple Proof

\[
\begin{align*}
\Gamma, A & \Rightarrow B, A, \Delta & \Gamma, A & \Rightarrow B, \Delta \\
\Gamma, A, (A \rightarrow B) & \Rightarrow B, \Delta & \Gamma, (A \& (A \rightarrow B)) & \Rightarrow B, \Delta \\
\Gamma & \Rightarrow (A \& (A \rightarrow B)) \rightarrow B, \Delta
\end{align*}
\]

A proof is **closed**, if all its branches are closed.
Sequent Calculus for FOL

- $\{t/t'\} \phi$ is result of replacing each occurrence of $t$ in $\phi$ with $t'$
- $t^{z'}$ any variable free term of type $z' \prec z$
- $c^z$ **new** constant of type $z$ (occurs not in current proof branch)
- Equations can be reversed by commutativity
Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>left side, antecedent</th>
<th>right side, succedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma, \forall z , x; \phi, { x/t' } \phi \Rightarrow \Delta ]</td>
<td>[ \Gamma \Rightarrow { x/c^z } \phi, \Delta ]</td>
</tr>
<tr>
<td>[ \Gamma, \forall z , x; \phi \Rightarrow \Delta ]</td>
<td>[ \Gamma \Rightarrow \forall z , x; \phi, \Delta ]</td>
</tr>
</tbody>
</table>

- \( \{ t/t' \} \phi \) is result of replacing each occurrence of \( t \) in \( \phi \) with \( t' \)
- \( t^{z'} \) any variable free term of type \( z' \prec z \)
- \( c^z \) new constant of type \( z \) (occurs not in current proof branch)
- Equations can be reversed by commutativity
Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>left side, antecedent</th>
<th>right side, succedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma, \forall z x; \phi, {x/t^z'} \phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \forall z x; \phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma \Rightarrow \forall z x; \phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \forall z x; \phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, {x/c^z} \phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow {x/t^z'} \phi, \exists z x; \phi \Rightarrow \Delta )</td>
</tr>
<tr>
<td>( \Gamma \Rightarrow \exists z x; \phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \exists z x; \phi \Rightarrow \Delta )</td>
</tr>
</tbody>
</table>

- \( \{t/t'\} \phi \) is result of replacing each occurrence of \( t \) in \( \phi \) with \( t' \)
- \( t^z' \) any variable free term of type \( z' \prec z \)
- \( c^z \) new constant of type \( z \) (occurs not in current proof branch)
- Equations can be reversed by commutativity
Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>left side, antecedent</th>
<th>right side, succedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma, \forall z \ x; \phi, {x/tz} \phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow {x/cz} \phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, \forall z \ x; \phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \forall z \ x; \phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, {x/cz} \phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow {x/tz} \phi, \exists z \ x ; \phi )</td>
</tr>
<tr>
<td>( \Gamma, t_1 \equiv t_2, {t_1/t_2} \psi \Rightarrow {t_1/t_2} \phi, \Delta )</td>
<td>( \Gamma \Rightarrow t \equiv t, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, t_1 \equiv t_2, \psi \Rightarrow \phi, \Delta )</td>
<td></td>
</tr>
</tbody>
</table>

- \( \{t/t'\} \phi \) is result of replacing each occurrence of \( t \) in \( \phi \) with \( t' \)
- \( tz' \) any variable free term of type \( z' \prec z \)
- \( cz \) new constant of type \( z \) (occurs not in current proof branch)
- Equations can be reversed by commutativity
Some Predefined Symbols in KeY Logic

Types
int, boolean, classes of the Java context of the proof obligation

Predicates on int
>, <, >=, <=

Functions and Constants
‘+’, ‘-’, ‘/’, ‘%’, ‘0’, ‘1’, ...
‘TRUE’, ‘FALSE’
\assumes (seq) \find (\vdash_{opt} \Phi) \replacewith(\vdash_{opt} \Phi') \add(\vdash \text{seq}') \ldots; \ldots; \ldots \\
heuristics(name^+)}
\textbf{Taclets - The rule description language of KeY}

\begin{align*}
\text{\textbackslash assumes} & (seq) & \text{\textbackslash find} & (\vdash_{opt} \Phi) & \text{\textbackslash replacewith} & (\vdash_{opt} \Phi') \\
& & & & \text{\textbackslash add} & (\vdash seq') \ldots; \ldots; \\
& & & & \text{\textbackslash heuristics} & (name^+) \\
\end{align*}

**Syntax**
\begin{itemize}
\item \texttt{\textbackslash find} \quad \text{sequent (max. one formula), formula or term}
\item \texttt{\textbackslash assumes} \quad \text{additional condition}
\item \texttt{\textbackslash replacewith} \quad \text{replaces the \texttt{\textbackslash find} part (\vdash_{opt} \text{depends on \texttt{\textbackslash find}})}
\item \texttt{\textbackslash add} \quad \text{adds the sequent to the antecedent or succedent}
\item ; \quad \text{start new subgoal}
\item \texttt{\textbackslash heuristics} \quad \text{adds the taclet to the enumerated heuristics}
\end{itemize}
The and-right rule as taclet

\[
\text{TEXTBOOK} \\
\frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \land B, \Delta} \quad (\text{and} - \text{right})
\]
The and-right rule as taclet

**Textbook**

\[
\frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \land B, \Delta} \quad (\text{and } - \text{ right})
\]

**Taclet**

\[
\text{\texttt{\textbackslash find}( \vdash A \land B )} \\
\text{\texttt{\textbackslash replacewith}( \vdash A );} \\
\text{\texttt{\textbackslash replacewith}( \vdash B );} \\
\text{\texttt{\textbackslash heuristics}(simplify)}
\]
\sorts \{ // types are called 'sorts'
    person; // one declaration per line, end with ‘;’
\}
\functions \{ // ResultType FctSymbol(ParType,..,ParType)
    int age(person); // 'int' predefined type
\}
\predicates \{ // PredSymbol(ParType,..,ParType)
    parent(person,person);
\}
\problem \{ // Goal formula, // ’>=’ predef.
    \forall person son; \forall person father; ( 
        parent(father,son) -> age(father) >= age(son))
\}
Another Example

Types \( T = \{ z \} \)

Predicates \( \mathcal{P} = \{ p \}, \sigma(p) = \langle z, z \rangle \)

Functions \( \mathcal{F} = \{ \} \)

\( (\exists z \ x; \exists z \ y; p(x, y) \land \forall z \ x; \neg p(x, x)) \rightarrow \exists z \ x; \exists z \ y; (\neg x \equiv y) \)

Intuitive Meaning? Satisfiable? Valid?

Demo

demo1.key
Types?
Types?
Library, Person, Student, Professor (+ some predefined)

Functions?

**Example: JML to FOL**

Types? Library, Person, Student, Professor (+ some predefined)

Functions?  

**Attributes**
- int Person.duration
- int Student.matnr

**Queries**
- String Professor.getInstitute
  - incl. some predefined
Example: JML to FOL

Meaning?

```java
public class Student {
    /** @ public invariant (\forall Student s; s.matnr==matnr; s==this); @ */
}
```
A student is uniquely identified by his/her student id (matnr)

```java
public class Student {
    /*@ public invariant (\forall Student s; s.matnr==matnr; s==this);@*/ ..
}
```

in FOL?
A student is uniquely identified by his/her student id (matnr)

```java
public class Student {
    /*@ public invariant (\forall Student s; s.matnr = matnr; s = this);@*/ ..
}
```

In FOL:

```
\forall Student p1; \forall Student p2;
(p1.matnr = p2.matnr -> p1 = p2)
```
“There is a tradition in logic, carried over into computer science, to think of pure first order logic as a universal language. In fact first order language is about as useful in verification as a Turing machine is in software engineering:

CUTE TO WATCH BUT NOT VERY USEFUL.”

V. Pratt
(Closed) FOL formula is either true or false wrt interpretation $\mathcal{D}$

Consider $\mathcal{D} = (U, I)$ to be static part of snapshot, ie state

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

Execution of program $p$ may change state, ie value of $x$
(Closed) FOL formula is either true or false wrt interpretation $\mathcal{D}$
Consider $\mathcal{D} = (U, I)$ to be static part of snapshot, ie state

Let $x$ be program (local) variable or attribute
Execution of program $p$ may change state, ie value of $x$

**Example**
Executing $x = 3$ results in $\mathcal{D}$ such that $\mathcal{D} \models x \doteq 3$
Executing $x = 4$ results in $\mathcal{D}$ such that $\mathcal{D} \nolhd x \doteq 3$
State Dependency of Formula Evaluation

(Closed) FOL formula is either true or false wrt interpretation $\mathcal{D}$

Consider $\mathcal{D} = (U, I)$ to be static part of snapshot, ie state

Let $x$ be program (local) variable or attribute
Execution of program $p$ may change state, ie value of $x$

Example
Executing $x = 3$ results in $\mathcal{D}$ such that $\mathcal{D} \models x \doteq 3$
Executing $x = 4$ results in $\mathcal{D}$ such that $\mathcal{D} \not\models x \doteq 3$

Need a logic to capture state before/after program execution
Definition (Signature)

\[ \Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \mathcal{PV}, \alpha, \sigma, \Pi_0, \mathcal{O} \cup \mathcal{Q} \cup \{ \vdash, \langle \cdot \rangle, [\cdot] \cdot \}) \]

**Type Symbols**  \( \mathcal{T} = \{ \text{int, boolean} \} \)

**Logical Variables**  \( \mathcal{V} = \{ y_i \mid i \in \mathbb{N} \} \)

**Predicate Symbols**  \( \mathcal{P} = \{ >, > =, <, <= \} \)

**Function Symbols**  \( \mathcal{F} = \{ +, -, *, 0, 1, \ldots \} \)

**Program Variables**  \( \mathcal{PV} = \{ x_i \mid i \in \mathbb{N} \} \)

Signature of functions/predicates as usual
Dynamic Logic (Simple Version) Signature

**Definition (Signature)**

\[ \Sigma = (T, V, P, F, PV, \alpha, \sigma, \Pi_0, O \cup Q \cup \{=, \langle \cdot \rangle \cdot, [\cdot] \cdot \}) \]

**Type Symbols**  \[ T = \{ \text{int, boolean} \} \]

**Logical Variables**  \[ V = \{ y_i \mid i \in \mathbb{N} \} \]

**Predicate Symbols**  \[ P = \{ >, \geq, <, \leq \} \]

**Function Symbols**  \[ F = \{ +, -, \times, 0, 1, \ldots \} \]

**Program Variables**  \[ PV = \{ x_i \mid i \in \mathbb{N} \} \]

Signature of functions/predicates as usual

**Atomic Programs** \( \Pi_0 \):

**Assignments**  \[ x = t \] with \( x \in PV \), \( t \) term of type int w/o logical variables
Definition (Signature)

\[ \Sigma = (T, V, P, F, PV, \alpha, \sigma, \Pi_0, O \cup Q \cup \{=, \langle \cdot \rangle \cdot, [\cdot] \cdot\}) \]

**Type Symbols** \( T = \{ \text{int}, \text{boolean} \} \)

**Logical Variables** \( V = \{ y_i \mid i \in N \} \)

**Predicate Symbols** \( P = \{ >, >=, <, <= \} \)

**Function Symbols** \( F = \{ +, -, *, 0, 1, \ldots \} \)

**Program Variables** \( PV = \{ x_i \mid i \in N \} \)

signature of functions/predicates as usual

**Atomic Programs** \( \Pi_0 \):

**Assignments** \( x = t \) with \( x \in PV, t \) term of type int w/o logical variables

**Modal Connectives** \( \langle \cdot \rangle \cdot \) “diamond”, \( [\cdot] \cdot \) “box”

First argument program, second argument formula
Programs $\Pi$

- If $\pi$ is an atomic program, then $\pi;$ is a program
Programs $\Pi$

- If $\pi$ is an atomic program, then $\pi; \alpha$ is a program
- If $\alpha$ and $\gamma$ are programs, then $\alpha; \gamma$ is a program
Dynamic Logic (Simple Version) Programs

**Programs \( \Pi \)**

- If \( \pi \) is an atomic program, then \( \pi; \) is a program
- If \( \alpha \) and \( \gamma \) are programs, then \( \alpha\gamma \) is a program
- If \( b \) is a variable-free term of type boolean, \( \alpha \) and \( \gamma \) programs, then

  \[
  \text{if } (b) \{ \alpha \} \text{ else } \{ \gamma \};
  \]

  is a program
Programs $\Pi$

- If $\pi$ is an atomic program, then $\pi;\,\pi$ is a program
- If $\alpha$ and $\gamma$ are programs, then $\alpha\gamma$ is a program
- If $b$ is a variable-free term of type $\text{boolean}$, $\alpha$ and $\gamma$ programs, then

\[
\text{if} \ (b) \ \{\alpha\} \ \text{else} \ \{\gamma\};
\]

is a program
- If $b$ is a variable-free term of type $\text{boolean}$, $\alpha$ a program, then

\[
\text{while} \ (b) \ \{\alpha\};
\]

is a program
Dynamic Logic Syntax Example

An admissible DL program $\alpha$:

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

What does $\alpha$ compute?
Dynamic Logic (Simple Version) Terms

Terms
Defined as in FOL using also $\mathcal{PV}$, but:

**Rigid versus Flexible**

- **rigid** symbols, same interpretation in all execution states
  - Needed, for example, to hold initial value of program variable
  - Logical variables and predefined functions/predicates are rigid

- **non-rigid** (or **flexible**) terms, interpretation depends on state
  - Needed to capture state change after program execution
  - Program variables are flexible

A term containing at least one flexible symbol is **flexible**, otherwise **rigid**
Dynamic Logic Formulas (DL Formulas)

- Each FOL formula is a DL formula
  - DL formulas closed under FOL operators and connectives, **but**
  - Program variables are never bound in quantifiers

- If $\alpha$ is a program and $\phi$ a DL formula then
  $\langle \alpha \rangle \phi$ is a DL formula
  $[\alpha] \phi$ is a DL-Formula

Programs contain no logical variables
Modalities can be arbitrarily nested
\forall int \ y; ((\langle x = 1; \rangle x \div y) \leftrightarrow (\langle x = 1 \ast 1; \rangle x \div y)) \quad Syntax ?
forall int y; ((x = 1; x \div y) <-> (x = 1 \ast 1; x \div y))
Dynamic Logic Syntax Example

\[ \forall \text{int } y; (⟨x = 1;⟩ x \downarrow y) \leftrightarrow (x = 1 \ast 1; x \downarrow y) \]

\[ \exists \text{int } x; ([x = 1;] (x \downarrow 1)) \]

Syntax ?
forall int y; (\(\langle x = 1; \rangle x \vdash y \rangle \leftrightarrow (x = 1 \ast 1; \rangle x \vdash y)) \quad \text{ok}

exists int x; ([x = 1;] (x \vdash 1)) \quad \text{bad}

- $x$ cannot be \textbf{logical variable}, because it occurs in program
- $x$ cannot be \textbf{program variable}, because it is quantified
\forall \text{int } y; ((x = 1; x \downarrow y) \leftrightarrow (x = 1 \ast 1; x \downarrow y)) \quad \text{ok}

\exists \text{int } x; ([x = 1;] (x \downarrow 1)) \quad \text{bad}

- x cannot be \textbf{logical variable}, because it occurs in program
- x cannot be \textbf{program variable}, because it is quantified

\langle x = 1; \rangle ([\text{while} (\text{true}) \{\} \] false) \quad \text{Syntax ?}
Dynamic Logic Syntax Example

\forall int y; ((x = 1; x ÷ y) <-> (x = 1 * 1; x ÷ y)) \quad \text{ok}

\exists int x; ([x = 1;] (x ÷ 1)) \quad \text{bad}

- x cannot be \text{logical variable}, because it occurs in program
- x cannot be \text{program variable}, because it is quantified

\langle x = 1; \rangle ([\text{while (true) \{\}}] \text{false}) \quad \text{ok}

- Program formulas can appear nested
A Kripke structure $K = (S, \rho)$ where
- $s = (U, I) \in S$ is a State/Interpretation and
- $\rho : \Pi \rightarrow (S \rightarrow S)$ $\rho(\alpha), \rho(\gamma)$ an admissible relation

Each state is first-order interpretation
Definition (Program Formulas)

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

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

- $s, \beta \models [\alpha] \phi$ iff $\rho(\alpha)(s), \beta \models \phi$ whenever $\rho(\alpha)(s)$ defined
  - If $\alpha$ terminates then $\phi$ is true in the final state after execution
Program Correctness

- $s, \beta \models \langle \alpha \rangle \phi$
  - $\alpha$ totally correct (with respect to $\phi$) in $s, \beta$
Program Correctness

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

- $s, \beta \models [\alpha] \phi$
  - $\alpha$ partially correct (with respect to $\phi$) in $s, \beta$
Program Correctness

- $s, \beta \models \langle \alpha \rangle \phi$
  - $\alpha$ totally correct (with respect to $\phi$) in $s, \beta$
- $s, \beta \models [\alpha] \phi$
  - $\alpha$ partially correct (with respect to $\phi$) in $s, \beta$
- Duality $\langle \alpha \rangle \phi \iff ! [\alpha] ! \phi$

Exercise: justify this with semantic definitions
Program Correctness

- $s, \beta \models \langle \alpha \rangle \phi$
  - $\alpha$ totally correct (with respect to $\phi$) in $s, \beta$
- $s, \beta \models [\alpha] \phi$
  - $\alpha$ partially correct (with respect to $\phi$) in $s, \beta$
- Duality $\langle \alpha \rangle \phi$ iff $! [\alpha] ! \phi$
  - Exercise: justify this with semantic definitions
- Implication if $\langle \alpha \rangle \phi$ then $[\alpha] \phi$
Validity of DL sequents compatible validity FOL sequents

\[ \Gamma \Rightarrow \Delta \text{ is valid iff it is true in all states } s \text{ in all Kripke structures } K \]
Semantics of Sequents

Validity of DL sequents compatible validity FOL sequents
\( \Gamma \Rightarrow \Delta \) is valid iff it is true in all states \( s \) in all Kripke structures \( K \)

How to restrict validity to set of initial states \( J \subseteq S \) ?

1. Design closed FOL formula \( \text{Init} \) with
   \[ s \models \text{Init} \iff s \in J \]

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

Later: simple method for specifying initial value of program variables
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$
Predicate symbols (prop. vars.) \( \mathcal{P} = \{a, b, c\} \)

\[
\begin{align*}
\text{s}_1 & \quad |\quad (a, b) \quad |\quad (\alpha) \quad a \\
\text{s}_2 & \quad |\quad (c) \quad |\quad (\gamma) \quad a \\
\text{s}_3 & \quad |\quad (\gamma) \quad a \\
\text{s}_4 & \quad |\quad (\alpha) \quad a \\
\text{s}_5 & \quad |\quad (\alpha) \quad a \\
\text{s}_6 & \quad |\quad (\gamma) \quad a
\end{align*}
\]
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$

\[ s_1 \models \langle \alpha \rangle a \text{ (ok)}, \]
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$

\[
\begin{align*}
  s_1 & \models \langle \alpha \rangle a \text{ (ok)}, \\
  s_1 & \models \langle \gamma \rangle a \text{ ?}
\end{align*}
\]
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$

$s_1 \models \langle \alpha \rangle a$ (ok), $s_1 \models \langle \gamma \rangle a$ (—)
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$

$s_1 \models \langle \alpha \rangle a$ (ok), \quad s_1 \models \langle \gamma \rangle a$ (—)

$s_5 \models \langle \gamma \rangle a$ ?
Predicate symbols (prop. vars.) \( \mathcal{P} = \{a, b, c\} \)

\[
\begin{align*}
\vdash (\alpha) a \quad & \text{(ok)}, \\
\vdash (\gamma) a \quad & \text{(-)}
\end{align*}
\]
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$

$s_1 |\models \langle \alpha \rangle a \ (\text{ok}), \quad s_1 |\models \langle \gamma \rangle a \ (\text{—})$

$s_5 |\models \langle \gamma \rangle a \ (\text{—}), \quad s_5 |\models [\gamma] a \ ?$
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$

- $s_1 \models \langle \alpha \rangle a \ (\text{ok})$
- $s_1 \models \langle \gamma \rangle a \ (-)$
- $s_5 \models \langle \gamma \rangle a \ (-)$
- $s_5 \models [\gamma] a \ (\text{ok})$
States $s = (U, I)$ have all the same universe $U$
May assume $\rho(\alpha)$ works on interpretations $I$
Define $I, \beta \models \phi$ as $s, \beta \models \phi$, where $s = (U, I)$
States $s = (U, I)$ have all the same universe $U$
May assume $\rho(\alpha)$ works on interpretations $I$
Define $I, \beta \models \phi$ as $s, \beta \models \phi$, where $s = (U, I)$
Program variables are flexible
Consider program variables as flexible constants in $s$ with value $I(x)$
States $s = (U, I)$ have all the same universe $U$
May assume $\rho(\alpha)$ works on interpretations $I$
Define $I, \beta \models \phi$ as $s, \beta \models \phi$, where $s = (U, I)$

Program variables are flexible
Consider program variables as flexible constants in $s$ with value $I(x)$

**State update** (cf. updated variable assignment) of $I$ at $y$ with $d \in U$

$$I_y^d(x) = \begin{cases} 
I(x) & x \neq y \\
 d & x = y 
\end{cases}$$
Operational Semantics of Programs

State transformation $\rho$ defines **semantics of programs**

Same $\rho$ for all programs, so not part of $s$; given $\beta$

- $\rho(x = t;)(I) = I_x^{t,\beta}$
State transformation $\rho$ defines **semantics of programs**

Same $\rho$ for all programs, so not part of $s$; given $\beta$

- $\rho(x = t;)(l) = l_x^{t,\beta}$
- $\rho(\text{if } (b) \{\alpha\} \text{ else } \{\gamma\};)(l) = \begin{cases} 
\rho(\alpha)(l) & l, \beta \models b \models \text{TRUE} \\
\rho(\gamma)(l) & \text{otherwise}
\end{cases}$
State transformation $\rho$ defines **semantics of programs**
Same $\rho$ for all programs, so not part of $s$; given $\beta$

- $\rho(x = t;)(I) = I_x^{t,\beta}$
- $\rho(\text{if } (b) \{\alpha\} \text{ else } \{\gamma\};)(I) = \begin{cases} 
\rho(\alpha)(I) & I, \beta \models b \models \text{TRUE} \\
\rho(\gamma)(I) & \text{otherwise}
\end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, undefined otherwise
Operational Semantics of Programs

State transformation $\rho$ defines **semantics of programs**
Same $\rho$ for all programs, so not part of $s$; given $\beta$

- $\rho(x = t;)(I) = I_x^{t,\beta}$
- $\rho(\text{if } (b) \{\alpha\} \text{ else } \{\gamma\};)(I) = \begin{cases} 
\rho(\alpha)(I) & I, \beta \models b \equiv \text{TRUE} \\
\rho(\gamma)(I) & \text{otherwise} 
\end{cases}$
- $\rho(\alpha \gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, undefined otherwise
- $\rho(\text{while } (b) \{\alpha\};)(I) = l'$ iff there are $l = I_0, \ldots, I_n = l'$ such that
Operational Semantics of Programs

State transformation $\rho$ defines **semantics of programs**
Same $\rho$ for all programs, so not part of $s$; given $\beta$

- $\rho(x = t;)(I) = I_x^{t,\beta}$
- $\rho(\text{if } (b) \{\alpha\} \text{ else } \{\gamma\};)(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \doteq \text{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha \gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, undefined otherwise
- $\rho(\text{while } (b) \{\alpha\};)(I) = I'$ iff there are $I = I_0, \ldots, I_n = I'$ such that
  - $I_j, \beta \models b \doteq \text{TRUE}$ for $0 \leq j < n$
State transformation $\rho$ defines **semantics of programs**

Same $\rho$ for all programs, so not part of $s$; given $\beta$

- $\rho(x = t;)(I) = l^t_{x,\beta}$
- $\rho(\text{if } (b) \{\alpha\} \text{ else } \{\gamma\};)(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \models \text{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, **undefined** otherwise
- $\rho(\text{while } (b) \{\alpha\};)(I) = l'$ iff there are $l = l_0, \ldots, l_n = l'$ such that
  - $l_j, \beta \models b \models \text{TRUE}$ for $0 \leq j < n$
  - $\rho(\alpha)(l_j) = l_{j+1}$ for $0 \leq j < n$
Operational Semantics of Programs

State transformation $\rho$ defines **semantics of programs**
Same $\rho$ for all programs, so not part of $s$; given $\beta$

- $\rho(x = t;)(I) = I_{x}^{t,\beta}$
- $\rho(\text{if } (b) \{ \alpha \} \text{ else } \{ \gamma \};)(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \equiv \text{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, undefined otherwise
- $\rho(\text{while } (b) \{ \alpha \};)(I) = I'$ iff there are $I = I_{0}, \ldots, I_{n} = I'$ such that
  - $I_{j}, \beta \models b \equiv \text{TRUE}$ for $0 \leq j < n$
  - $\rho(\alpha)(I_{j}) = I_{j+1}$ for $0 \leq j < n$
  - $I_{n}, \beta \models b \equiv \text{FALSE}$ undefined otherwise
Dynamic Logic Examples

Partial correctness assertion  (Hoare formula)

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

If \alpha is started in a state satisfying \psi and terminates, then its final state satisfies \phi

In DL

\psi \rightarrow [\alpha] \phi
Partial correctness assertion (Hoare formula)

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

If \(\alpha\) is started in a state satisfying \(\psi\) and terminates, then its final state satisfies \(\phi\)

In DL

\(\psi \rightarrow [\alpha]\phi\)

Valid formulas

\([x = 1;] (x \div 1)\)
Partial correctness assertion (Hoare formula)

\[ \{\psi\} \alpha \{\phi\} \]

If \( \alpha \) is started in a state satisfying \( \psi \) and terminates, then its final state satisfies \( \phi \)

In DL

\( \psi \rightarrow [\alpha] \phi \)

Valid formulas

\[ [x = 1;] (x \div 1) \]

\[ [\text{while (true) } \{x = x;\}; ] \text{false} \]
Dynamic Logic Examples

Partial correctness assertion (Hoare formula)

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

If \alpha is started in a state satisfying \psi and terminates, then its final state satisfies \phi

In DL \psi \rightarrow [\alpha] \phi

Valid formulas

[x = 1;] (x \div 1)

[while (true) \{x = x;\}; ] false

Validity depends on \alpha, \gamma

\forall int y; (\langle \alpha \rangle x \div y) \leftrightarrow (\langle \gamma \rangle x \div y)

meaning?
Dynamic Logic Examples

Partial correctness assertion (Hoare formula)

\[ \{\psi\} \alpha \{\phi\} \]

If \( \alpha \) is started in a state satisfying \( \psi \) and terminates, then its final state satisfies \( \phi \)

In DL

\[ \psi \rightarrow [\alpha] \phi \]

Valid formulas

\[ [x = 1;] (x \div 1) \quad [\text{while (true) } \{x = x;\};] \text{ false} \]

Validity depends on \( \alpha, \gamma \)

\[ \forall \text{ int } y; (\langle \alpha \rangle x \div y) \leftrightarrow (\langle \gamma \rangle x \div y)) \quad \alpha, \gamma \text{ equiv. relative to } x \]
Need to have rules for program formulas: but which?
What corresponds to top-level connective in **sequential** program?
Need to have rules for program formulas: but which?
What corresponds to top-level connective in **sequential** program?

**Idea:** follow natural program control flow
Need to have rules for program formulas: but which?
What corresponds to top-level connective in *sequential* program?

**Idea**: follow natural program control flow

Sound and complete rule for conclusions with main formulas:

\[
\langle \xi \gamma \rangle \phi, \quad [\xi \gamma] \phi
\]

where $\xi$ one *single* admissible program statement
Proof by Symbolic Program Execution

Need to have rules for program formulas: but which?
What corresponds to top-level connective in sequential program?

**Idea:** follow natural program control flow

Sound and complete rule for conclusions with main formulas:

\[ \langle \xi \gamma \rangle \phi, \quad [\xi \gamma] \phi \]

where \( \xi \) one **single** admissible program statement

Rules **execute symbolically** the first active statement
Proof corresponds to symbolic program execution
Dynamic Logic Calculus

\[
\text{CONCATENATE} \quad \frac{\Gamma \implies (\langle \alpha \rangle (\langle \gamma \rangle \phi), \Delta)}{\Gamma \implies (\langle \alpha \gamma \rangle \phi, \Delta)}
\]
Dynamic Logic Calculus

\[
\text{CONCATENATE:} \quad \Gamma \implies (\langle \alpha \rangle (\langle \gamma \rangle \phi), \Delta) \\
\Gamma \implies (\langle \alpha \gamma \rangle \phi, \Delta)
\]

\[
\text{IF:} \quad \Gamma, b \doteq \text{TRUE} \implies (\langle \alpha \rangle \phi, \Delta) \\
\Gamma, b \doteq \text{FALSE} \implies (\langle \gamma \rangle \phi, \Delta)
\]

\[
\Gamma \implies (\text{if (b) \{\alpha\} else \{\gamma\};}) \phi, \Delta
\]
Dynamic Logic Calculus

**CONCATENATE**

\[
\frac{\Gamma \Rightarrow \langle \alpha \rangle \langle \gamma \rangle \phi, \Delta}{\Gamma \Rightarrow \langle \alpha \gamma \rangle \phi, \Delta}
\]

**IF**

\[
\frac{\Gamma, b \models \text{TRUE} \Rightarrow \langle \alpha \rangle \phi, \Delta \quad \Gamma, b \models \text{FALSE} \Rightarrow \langle \gamma \rangle \phi, \Delta}{\Gamma \Rightarrow \langle \text{if} \ (b) \ \{\alpha\} \ \text{else} \ \{\gamma\};\rangle \phi, \Delta}
\]

**UNWIND**

\[
\frac{\Gamma, b \models \text{FALSE} \Rightarrow \phi, \Delta \quad \Gamma, b \models \text{FALSE} \Rightarrow \langle \alpha \rangle \text{while} \ (b) \ \{\alpha\};\rangle \phi, \Delta}{\Gamma \Rightarrow \langle \text{while} \ (b) \ \{\alpha\};\rangle \phi, \Delta}
\]
**Dynamic Logic Calculus**

\[
\text{CONCATENATE} \quad \frac{\Gamma \implies \langle \alpha \rangle (\langle \gamma \rangle \phi), \Delta}{\Gamma \implies \langle \alpha \gamma \rangle \phi, \Delta}
\]

\[
\text{IF} \quad \frac{\Gamma, b \models \text{TRUE} \implies \langle \alpha \rangle \phi, \Delta}{\Gamma \implies \langle \text{if} (b) \{\alpha\} \text{ else } \{\gamma\}; \rangle \phi, \Delta}
\]

\[
\text{UNWIND} \quad \frac{\Gamma, b \models \text{FALSE} \implies \phi, \Delta}{\Gamma \implies \langle \text{while} (b) \{\alpha\}; \rangle \phi, \Delta}
\]

\[
\text{ASSIGNMENT} \quad \frac{\Gamma^{x/x'}, x \models t \implies \phi, \Delta^{x/x'}}{\Gamma \implies \langle x = t; \rangle \phi, \Delta}
\]
Assignment Rule Using Updates

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

Avoids renaming of program variables

**But:** rules dealing with programs need to account for updates
Assignment Rule Using Updates

\[
\frac{\Gamma \implies \{x := t\} \phi, \Delta}{\Gamma \implies \langle x = t; \rangle \phi, \Delta}
\]

Avoids renaming of program variables

**But:** rules dealing with programs need to account for updates

Rules work on **first active statement** after **prefix**, followed by **postfix** (remaining code)

Explicit concatenation rule not longer needed
Assignment Rule Using Updates

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

Avoids renaming of program variables

**But:** rules dealing with programs need to account for updates

Rules work on **first active statement** after *prefix*, followed by *postfix* (remaining code)

Explicit concatenation rule not longer needed

**General form of conclusion in rule for symbolic execution**

\[
\Gamma \Rightarrow \langle \pi \xi; \omega \rangle \phi, \Delta
\]
Explicit State Updates

**Updates** record state change
Updates record state change

Syntax
If \( v \) is program variable, \( t, t' \) terms, and \( \phi \) any DL formula, then \( \{v := t\}\phi \) is DL formula and \( \{v := t\}t' \) is term
Updates record state change

Syntax
If \( v \) is program variable, \( t, t' \) terms, and \( \phi \) any DL formula, then \( \{ v := t \} \phi \) is DL formula and \( \{ v := t \} t' \) is term

Semantics
\( I, \beta \models \{ v := t \} \phi \) iff \( I^{t} \beta, \beta \models \phi \)

Semantics identical to assignment

Updates work as “lazy” assignments
Update followed by \textit{program variable}

\[ \{x := t\}y \leadsto y \]
\[ \{x := t\}x \leadsto t \]
Computing Effect of Updates

Update followed by **program variable**
\[
\{x := t\}y \leadsto y
\]
\[
\{x := t\}x \leadsto t
\]

Update followed by **complex term**
\[
\{x := t\}f(t_1, \ldots, t_n) \leadsto f(\{x := t\}t_1, \ldots, \{x := t\}t_n)
\]
Computing Effect of Updates

Update followed by **program variable**
\[ \{ x := t \} y \leadsto y \]
\[ \{ x := t \} x \leadsto t \]

Update followed by **complex term**
\[ \{ x := t \} f(t_1, \ldots, t_n) \leadsto f(\{ x := t \} t_1, \ldots, \{ x := t \} t_n) \]

Update followed by **first-order formula**
\[ \{ x := t \}(\phi \land \psi) \leadsto \{ x := t \} \phi \land \{ x := t \} \psi \]
\[ \{ x := t \}(\forall z y ; \phi) \leadsto \forall z y ; (\{ x := t \} \phi) \text{ etc.} \]
Computing Effect of Updates

Update followed by **program variable**
\[
\{x := t\}y \leadsto y
\]
\[
\{x := t\}x \leadsto t
\]

Update followed by **complex term**
\[
\{x := t\}f(t_1, \ldots, t_n) \leadsto f(\{x := t\}t_1, \ldots, \{x := t\}t_n)
\]

Update followed by **first-order formula**
\[
\{x := t\}(\phi \& \psi) \leadsto \{x := t\}\phi \& \{x := t\}\psi
\]
\[
\{x := t\}(\forall z y; \phi) \leadsto \forall z y; (\{x := t\}\phi) \text{ etc.}
\]

Update followed by **program formula**
\[
\{x := t\}(\langle \alpha \rangle \phi) \leadsto \{x := t\}(\langle \alpha \rangle \phi)
\]

**Update computation delayed until** \(\alpha\) **symbolically executed**
Example Proof

\programVariables { 
  int i; 
  int j; 
}
\problem { 
  \forall int x; \forall int y; 
  ( i=x & j=y -> 
    \langle\{int h = i; i = j; j = h;\}\rangle (i=y & j=x) ) 
}

Intuitive Meaning? Satisfiable? Valid?

Demo

\ilIntro/exchange.key