Entwicklung objektorientierter Software mit formalen Methoden

Program Verification – Dynamic Logic for Users

Bernhard Beckert

Universität Koblenz-Landau
Verification in different design phases

Analyse
Diagrams

+ 

Requirements
OCL + nat. Language

---
time
Verification in different design phases

Analyse
Diagrams

+ 

Requirements
OCL + nat. Language

(sematic gap)

time
Verification in different design phases

Analyse
Diagrams
+
Requirements
OCL + nat. Language

Design
Diagrams
+
Specification
OCL (inv., pre-/post)

(time)

(semantic gap)
Verification in different design phases

- **Analyse**
  - Diagrams
  - Requirements
    - OCL + nat. Language

- **Design**
  - Diagrams
  - Specification
    - OCL (inv., pre-/post)

Refinement

(semantic gap) Horizontal Verification

Praktikum WS0304: Formale Entwicklung objektorientierter Software — Introduction to OCL - p.2
Verification in different design phases

Analyse Diagrams

Design Diagrams

Implementation Diagrams

Requirements
OCL + nat. Language

Specification
OCL (inv., pre-/post)

Source Code
Java, C++, Prolog

Refinement
(time)

(semantic gap)

Horizontal Verification
Verification in different design phases

- **Analyse Diagrams**
  - Requirements
    - OCL + nat. Language

- **Design Diagrams**
  - Specification
    - OCL (inv., pre−/post)

- **Implementation Diagrams**
  - Source Code
    - Java, C++, Prolog

---

Refinement Equivalence

(time)

(specific gap)

---

Horizontal Verification

Vertical Verification

Praktikum WS0304: Formale Entwicklung objektorientierter Software — Introduction to OCL — p.2
What has to be proved?

Horizontal Verification

inv: $3 = 5$

Specification

Design by Contract
What has to be proved?

Horizontal Verification

- Consistency properties

Specification

inv. 3 = 5

Design by Contract
What has to be proved?

Horizontal Verification

- Consistency properties
- Compliance to design principles

Specification

Design by Contract

inv. 3 = 5
What has to be proved?

Horizontal Verification

- Consistency properties
- Compliance to design principles

⇒ source code is not involved
What has to be proved?

Horizontal Verification

- Consistency properties
- Compliance to design principles

⇒ source code is not involved

Horizontal Verification can be done in Classical First-Order Logic (FOL)

Specification

Design by Contract

inv. 3 = 5
Syntax of Propositional Logic

**Signature** \( \Sigma = (\mathcal{P}, \mathcal{O}) \)

- **Propositional Variables** \( \mathcal{P} = \{P_i \mid i \in \mathbb{IN}\} \)
Syntax of Propositional Logic

Signature $\Sigma = (\mathcal{P}, \mathcal{O})$

- Propositional Variables $\mathcal{P} = \{P_i | i \in \mathbb{N}\}$
- Logical Operators $\mathcal{O} = \{\land, \lor, \neg\}$ (handle $\rightarrow$, $\leftrightarrow$ as abbreviations)
Syntax of Propositional Logic

**Signature** \( \Sigma = (\mathcal{P}, \mathcal{O}) \)

- **Propositional Variables** \( \mathcal{P} = \{ P_i | i \in \mathbb{IN} \} \)
- **Logical Operators** \( \mathcal{O} = \{ \land, \lor, \neg \} \) (handle \( \rightarrow, \leftrightarrow \) as abbreviations)

**Formulas** \( For_0^\Sigma \)

- Propositional Variables are formulas
Syntax of Propositional Logic

**Signature** $\Sigma = (\mathcal{P}, \mathcal{O})$

- **Propositional Variables** $\mathcal{P} = \{P_i | i \in \mathbb{N}\}$
- **Logical Operators** $\mathcal{O} = \{\land, \lor, \lnot\}$ (handle $\to$, $\leftrightarrow$ as abbreviations)

**Formulas** $\mathcal{F}_{\Sigma}^0$

- Propositional Variables are formulas
- If $G$ and $H$ are formulas then

$$\lnot G, (G \land H) \text{ and } (G \lor H)$$

are also formulas
Interpretation (Assignment) $I$

Assigns a definite truth value to each propositional variable

$$I : \mathcal{P} \rightarrow \{ \text{true}, \text{false} \}$$
Semantics of Propositional Logic

Interpretation (Assignment) \(I\)

Assigns a definite truth value to each propositional variable

\[ I : \mathcal{P} \rightarrow \{true, false\} \]

Valuation \(val_I\): Continuation of \(I\) on \(For_0^{\Sigma}\)

\[ val_I : For_0^{\Sigma} \rightarrow \{true, false\} \]
Interpretation (Assignment) $I$

Assigns a definite truth value to each propositional variable

$$I : \mathcal{P} \rightarrow \{true, false\}$$

Valuation $val_I$: Continuation of $I$ on $For_0^\Sigma$

$$val_I : For_0^\Sigma \rightarrow \{true, false\}$$

$$val_I(P_i) = I(P_i)$$
Semantics of Propositional Logic

Interpretation (Assignment) $I$

Assigns a definite truth value to each propositional variable

$I : \mathcal{P} \rightarrow \{true, false\}$

Valuation $val_I$: Continuation of $I$ on $\text{For}^\Sigma_0$

$val_I : \text{For}^\Sigma_0 \rightarrow \{true, false\}$

\[
val_I(P_i) = I(P_i) \quad val_I(P_i \land P_j) = \begin{cases} 
  true & \text{if } val_I(P_i) = true \text{ and } val_I(P_j) = true \\
  false & \text{otherwise}
\end{cases}
\]

...(and so on)
»The truth that’s me.«, said the tautology.

Let $\Phi \in For_{0}^{\Sigma}$, $\Gamma \subset For_{0}^{\Sigma}$

- $I$ is a **model** for $\Phi$ iff. $val_I(\Phi) = true$ (write: $I \models \Phi$)
»The truth that’s me.«, said the tautology.

Let $\Phi \in For_0^\Sigma$, $\Gamma \subset For_0^\Sigma$

- $I$ is a model for $\Phi$ iff. $val_I(\Phi) = true$ (write: $I \models \Phi$)
- $\Gamma \models \Phi$ iff. for all interpretations $I$:
»The truth that’s me.«, said the tautology.

Let $\Phi \in For_0^\Sigma$, $\Gamma \subset For_0^\Sigma$

- $I$ is a model for $\Phi$ iff. $val_I(\Phi) = true$ (write: $I \models \Phi$)
- $\Gamma \models \Phi$ iff. for all interpretations $I$:

$$I \models \Psi \text{ for all } \Psi \in \Gamma \text{ then also } I \models \Phi$$
»The truth that’s me.«, said the tautology.

Let $\Phi \in For^\Sigma_0$, $\Gamma \subseteq For^\Sigma_0$

- *$I$ is a model for $\Phi$ iff. $val_I(\Phi) = true$* (write: $I \models \Phi$

- $\Gamma \models \Phi$ iff. for all interpretations $I$:
  $$I \models \Psi \text{ for all } \Psi \in \Gamma \text{ then also } I \models \Phi$$

- If $\Phi$ is valid under all interpretations, i.e
  $$\emptyset \models \Phi \text{ (short: } \models \Phi)$$

  then $\Phi$ is called a tautology.
THE SUN SHINES  THE PEOPLE ARE HAPPY
Orientation Map

Syntax

THE SUN SHINES

THE PEOPLE ARE HAPPY

A

B
**Orientation Map**

**Syntax**

\[
\text{IF \ THE \ SUN \ SHINES} \quad \rightarrow \quad \text{THE \ PEOPLE \ ARE \ HAPPY}
\]

**Example:**

\[
\text{IF \ THE \ SUN \ SHINES \ THEN \ THE \ PEOPLE \ ARE \ HAPPY}
\]
IF THE SUN SHINES THEN THE PEOPLE ARE HAPPY

Syntax

A → B

True

Semantics

False
IF THE SUN SHINES  THEN  THE PEOPLE ARE HAPPY

Syntax

A  \rightarrow  B

True  False

Now: Syntactical reasoning
If the sun shines then the people are happy

Syntax

A → B

True

Semantics

False

Now: Syntactical reasoning

A

The sun shines
IF THE SUN SHINES  THEN  THE PEOPLE ARE HAPPY

Syntax

A  →  B

True  Semantics  False

Now: Syntactical reasoning

A  THE SUN SHINES

A → B  IF THE SUN SHINES THEN THE PEOPLE ARE HAPPY.
Now: Syntactical reasoning

\[
\begin{align*}
A & \quad \text{THE SUN SHINES} \\
A \rightarrow B & \quad \text{IF THE SUN SHINES THEN THE PEOPLE ARE HAPPY.} \\
B & \quad \text{THE PEOPLE ARE HAPPY}
\end{align*}
\]
A Bridge between Semantics and Syntax

**Deduction Theorem**

Let $\Gamma \subseteq \text{For}_\Sigma$, $\Phi, \Psi \in \text{For}_\Sigma$

$$\Gamma, \Psi \models \Phi \iff \Gamma \models \Psi \rightarrow \Phi$$

Establishes a relationship between the semantical consequence '$\models$'
and the syntactical implication '$\rightarrow$'
Task: Compute $\Gamma \models \Phi$ by performing syntactical transformations
Reasoning as Syntactical Transformations

Task: Compute $\Gamma \models \Phi$ by performing syntactical transformations

Solution: Calculus $\vdash$ and a set of rules $\mathcal{R}$
Task: Compute $\Gamma \models \Phi$ by performing syntactical transformations

Solution: Calculus $\vdash$ and a set of rules $\mathcal{R}$

Sequent Calculus $\models$: 

$$\begin{array}{c}
\psi_1, \ldots, \psi_n \quad \Rightarrow \\
\text{Premises} \\
\phi_1, \ldots, \phi_n \quad \text{Consequences}
\end{array}$$
Reasoning as Syntactical Transformations

Task: Compute $\Gamma \models \Phi$ by performing syntactical transformations

Solution: Calculus $\vdash$ and a set of rules $\mathcal{R}$

Sequent Calculus $\iff$:

$$\psi_1, \ldots, \psi_n \Rightarrow \phi_1, \ldots, \phi_n$$

has the same semantic as

$$\psi_1 \wedge \ldots \wedge \psi_n \rightarrow \phi_1 \vee \ldots \vee \phi_n$$
## Rules of the Sequent Calculus

<table>
<thead>
<tr>
<th></th>
<th>left side</th>
<th>right side</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>
</tbody>
</table>
# Rules of the Sequent Calculus

<table>
<thead>
<tr>
<th>left side</th>
<th>right side</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>not</strong></td>
<td></td>
</tr>
<tr>
<td>[\Gamma \Rightarrow A, \Delta] \hline [\Gamma, \neg A \Rightarrow \Delta]</td>
<td></td>
</tr>
<tr>
<td><strong>and</strong></td>
<td></td>
</tr>
<tr>
<td>[\Gamma, A, B \Rightarrow \Delta] \hline [\Gamma, A \wedge B \Rightarrow \Delta]</td>
<td>[\Gamma \Rightarrow A, \Delta] \hline [\Gamma \Rightarrow B, \Delta] \hline [\Gamma \Rightarrow A \wedge B, \Delta]</td>
</tr>
</tbody>
</table>
### Rules of the Sequent Calculus

<table>
<thead>
<tr>
<th></th>
<th><strong>left side</strong></th>
<th></th>
<th><strong>right side</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>not</strong></td>
<td></td>
<td><strong>and</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \Gamma \implies A, \Delta )</td>
<td></td>
<td>( \Gamma, A \implies \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, \neg A \implies \Delta )</td>
<td></td>
<td>( \Gamma \implies \neg A, \Delta )</td>
</tr>
</tbody>
</table>
|          | \( \Gamma, A, B \implies \Delta \)               |          | \( \Gamma \implies A, \Delta \) \( \Gamma \implies B, \Delta \)
|          | \( \Gamma, A \land B \implies \Delta \)         |          | \( \Gamma \implies A \land B, \Delta \)          |
| **or**   | \( \Gamma, A \implies \Delta \) \( \Gamma, B \implies \Delta \) |          | \( \Gamma \implies A, B, \Delta \)               |
|          | \( \Gamma, A \lor B \implies \Delta \)          |          | \( \Gamma \implies A \lor B, \Delta \)           |
## Rules of the Sequent Calculus

<table>
<thead>
<tr>
<th></th>
<th>left side</th>
<th>right side</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>$\Gamma, A \land B \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow A \land B, \Delta$</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>$\Gamma, A \lor B \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow A \lor B, \Delta$</td>
</tr>
<tr>
<td>imp</td>
<td>$\Gamma \Rightarrow A, \Delta$ $\Gamma, B \Rightarrow \Delta$</td>
<td>$\Gamma, A \Rightarrow B, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, A \Rightarrow B, \Delta$</td>
<td>$\Gamma \Rightarrow A \Rightarrow B, \Delta$</td>
</tr>
</tbody>
</table>
### Rules of the Sequent Calculus

<table>
<thead>
<tr>
<th></th>
<th>Left Side</th>
<th>Right Side</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>$\Gamma \Rightarrow A \land B, \Delta$</td>
</tr>
<tr>
<td><strong>or</strong></td>
<td>$\Gamma, A \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow A, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, B \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow B, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, A \lor B \Rightarrow \Delta$</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></td>
</tr>
<tr>
<td></td>
<td>$\Gamma, A \rightarrow B \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow A \rightarrow B, \Delta$</td>
</tr>
</tbody>
</table>

**CLOSE(AXIOM)**

$\Gamma, \neg \neg A \Rightarrow \Delta$
Proof of Modus Ponens

\[ \Gamma \implies (A \land (A \rightarrow B)) \rightarrow B, \Delta \]
Proof of Modus Ponens

\[ \Gamma, (A \land (A \rightarrow B)) \quad \rightarrow \quad B, \Delta \]

\[ \Gamma \quad \rightarrow \quad (A \land (A \rightarrow B)) \rightarrow B, \Delta \]
Proof of Modus Ponens

\[
\begin{align*}
\Gamma, A, (A \rightarrow B) & \iff B, \Delta \\
\Gamma, (A \land (A \rightarrow B)) & \iff B, \Delta \\
\Gamma & \iff (A \land (A \rightarrow B)) \rightarrow B, \Delta
\end{align*}
\]
Proof of Modus Ponens

\[ \Gamma, A \implies B, A, \Delta \quad \Gamma, A, B \implies B, \Delta \]
\[ \Gamma, A, (A \implies B) \implies B, \Delta \]
\[ \Gamma, (A \land (A \implies B)) \implies B, \Delta \]
\[ \Gamma \implies (A \land (A \implies B)) \implies B, \Delta \]
**Proof of Modus Ponens**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma, A \implies B, A, \Delta )</td>
<td>( \Gamma, A, B \implies B, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, A, (A \rightarrow B) \implies B, \Delta )</td>
<td></td>
</tr>
<tr>
<td>( \Gamma, (A \land (A \rightarrow B)) \implies B, \Delta )</td>
<td></td>
</tr>
<tr>
<td>( \Gamma \implies (A \land (A \rightarrow B)) \rightarrow B, \Delta )</td>
<td></td>
</tr>
</tbody>
</table>
Proof of Modus Ponens

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

A proof is closed, if all its goals are closed.
Propositional logic is insufficient

\[ \forall \text{ PERSONS ARE HAPPY} \]
Propositional logic is insufficient

\[ A \quad \text{ALL PERSONS ARE HAPPY} \]

\[ B \quad \text{PAT IS A PERSON} \]
Propositional logic is insufficient

\[ A \quad \text{ALL PERSONS ARE HAPPY} \]
\[ B \quad \text{PAT IS A PERSON} \]
\[ ? \quad \text{PAT IS HAPPY} \]

Propositional Logic lacks a possibility to talk about individuals.
Propositional logic is insufficient

\[
\begin{align*}
A & \quad \text{ALL PERSONS ARE HAPPY} \\
B & \quad \text{PAT IS A PERSON} \\
? & \quad \text{PAT IS HAPPY}
\end{align*}
\]

Propositional Logic lacks a possibility to talk about individuals.

\[\Rightarrow \text{First-Order Logic (FOL)}\]
Syntax of First-Order Logic

**Signature** \( \Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{V}, \alpha, \mathcal{O} \cup Q \cup \{\doteq\}) \)
Syntax of First-Order Logic

**Signature** $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{V}, \alpha, \mathcal{O} \cup \mathcal{Q} \cup \{\div\})$

- **Predicate Symbols** $\mathcal{P} = \{P_i | i \in \mathbb{IN}\}$
- **Function Symbols** $\mathcal{F} = \{f_i | i \in \mathbb{IN}\}$
- **Variables** $\mathcal{V} = \{x_i | i \in \mathbb{IN}\}$

$\alpha : \mathcal{P} \cup \mathcal{F} \rightarrow \mathbb{IN} \text{ (arity)}$
Syntax of First-Order Logic

**Signature** $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{V}, \alpha, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

- **Predicate Symbols** $\mathcal{P} = \{P_i | i \in \mathbb{IN}\}$, $\alpha : \mathcal{P} \cup \mathcal{F} \rightarrow \mathbb{IN}$ (arity)
- **Function Symbols** $\mathcal{F} = \{f_i | i \in \mathbb{IN}\}$
- **Variables** $\mathcal{V} = \{x_i | i \in \mathbb{IN}\}$

- **Operators** $\mathcal{O} = \{\wedge, \vee, \neg\}$, **Quantifiers** $\mathcal{Q} = \{\forall, \exists\}$ and the syntactical equality $\doteq$
Syntax of First-Order Logic

**Signature** $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{V}, \alpha, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

- **Predicate Symbols** $\mathcal{P} = \{P_i | i \in \mathbb{IN}\}$, $\alpha : \mathcal{P} \cup \mathcal{F} \rightarrow \mathbb{IN}$ (arity)
- **Function Symbols** $\mathcal{F} = \{f_i | i \in \mathbb{IN}\}$
- **Variables** $\mathcal{V} = \{x_i | i \in \mathbb{IN}\}$

- **Operators** $\mathcal{O} = \{\land, \lor, \neg\}$, **Quantifiers** $\mathcal{Q} = \{\forall, \exists\}$ and the syntactical equality $\doteq$

Terms $\text{Term}_\Sigma$ and Formulas $\text{For}_\Sigma$ are defined inductively as usual.
Syntax of First-Order Logic

Signature $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{V}, \alpha, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

- **Predicate Symbols** $\mathcal{P} = \{P_i | i \in \mathbb{IN}\}$, with $\alpha : \mathcal{P} \cup \mathcal{F} \to \mathbb{IN}$ (arity)
- **Function Symbols** $\mathcal{F} = \{f_i | i \in \mathbb{IN}\}$
- **Variables** $\mathcal{V} = \{x_i | i \in \mathbb{IN}\}$

- **Operators** $\mathcal{O} = \{\land, \lor, \neg\}$, **Quantifiers** $\mathcal{Q} = \{\forall, \exists\}$ and the syntactical equality $\doteq$

Terms $\text{Term}_\Sigma$ and Formulas $\text{For}_\Sigma$ are defined inductively as usual.

**Additional:** Let $t_1, t_2$ be terms then $t_1 \doteq t_2$ is a formula.
Semantics of First-Order Logic

Interpretation \( \mathcal{D}=(U, I) \):

\( U \) is the non-empty universe
Interpretation $\mathcal{D}=(U, I)$:

$U$ is the non-empty universe

$P^I \subseteq \{(x_1, \ldots, x_n) | x_i \in U, n = \alpha(P)\}$

$f^I : U^\alpha(f) \rightarrow U$
Semantics of First-Order Logic

Interpretation $\mathcal{D}=(U, I)$:

$U$ is the non-empty universe

$P^I \subseteq \{(x_1, \ldots, x_n) | x_i \in U, n = \alpha(P)\}$

$f^I : U^\alpha(f) \to U$

Variable Assignment $\beta : \mathcal{V} \to U$

$\text{val}_{\mathcal{D}, \beta}(P(x_1, \ldots, x_n)) = \begin{cases} true & (\beta(x_1), \ldots, \beta(x_n)) \in P^I \\ false & otherwise \end{cases}$
Semantics of First-Order Logic

Interpretation $\mathcal{D} = (U, I)$:

$U$ is the non-empty universe

$P^I \subseteq \{(x_1, \ldots, x_n) | x_i \in U, n = \alpha(P)\}$

$f^I : U^{\alpha(f)} \rightarrow U$

Variable Assignment $\beta : \mathcal{V} \rightarrow U$

$\text{val}_{\mathcal{D}, \beta}(P(x_1, \ldots, x_n)) = \begin{cases} 
\text{true} & (\beta(x_1), \ldots, \beta(x_n)) \in P^I \\
\text{false} & \text{otherwise}
\end{cases}$

$\text{val}_{\mathcal{D}, \beta}(\forall x. \Phi(x)) = \begin{cases} 
\text{true} & \text{for all } d \in U : \text{val}_{\mathcal{D}, \beta_d}(\Phi) = \text{true} \\
\text{false} & \text{otherwise}
\end{cases}$
Definitions

Satisfiability, Model and Universal validity

\[ \mathcal{D}, \beta \models \Phi \iff \text{val}_{\mathcal{D}, \beta}(\Phi) = \text{true} \quad (\Phi \text{ is satisfiable}) \]
Definitions

Satisfiability, Model and Universal validity

\[ \mathcal{D}, \beta \models \Phi \iff \text{val}_{\mathcal{D}, \beta}(\Phi) = \text{true} \quad (\Phi \text{ is satisfiable}) \]

\[ \mathcal{D} \models \Phi \iff \text{for all } \beta : \mathcal{D}, \beta \models \Phi \quad (\Phi \text{ is valid}) \]
Definitions

Satisfiability, Model and Universal validity

\[ \mathcal{D}, \beta \models \Phi \quad \text{iff.} \quad \text{val}_{\mathcal{D}, \beta}(\Phi) = \text{true} \quad (\Phi \text{ is satisfiable}) \]

\[ \mathcal{D} \models \Phi \quad \text{iff.} \quad \text{for all } \beta : \mathcal{D}, \beta \models \Phi \quad (\Phi \text{ is valid}) \]

\[ \models \Phi \quad \text{iff.} \quad \text{for all } \mathcal{D} : \quad \mathcal{D} \models \Phi \quad (\Phi \text{ is universally valid}) \]
Definitions

Satisfiability, Model and Universal validity

\[ \mathcal{D}, \beta \models \Phi \iff \text{val}_{\mathcal{D}, \beta}(\Phi) = \text{true} \quad (\Phi \text{ is satisfiable}) \]

\[ \models \Phi \iff \text{for all } \beta : \mathcal{D}, \beta \models \Phi \quad (\Phi \text{ is valid}) \]

\[ \models \Phi \iff \text{for all } \mathcal{D} : \mathcal{D} \models \Phi \quad (\Phi \text{ is universally valid}) \]

Remark: Sorted First-Order Logic

Variables and functions is given a sort \( \in \text{Sorts} \)

\[
\forall x : S. \Phi(x) \quad \text{i.e.} \quad \forall x. (S(x) \rightarrow \Phi(x))
\]

\[
\exists x : S. \Phi(x) \quad \text{i.e.} \quad \exists x. (S(x) \land \Phi(x))
\]
Do we have a deduction theorem at hand?

\[ \Gamma, \Psi \models \Phi \iff \Gamma \models \Psi \to \Phi \]

?
Do we have a deduction theorem at hand?

\[ \Gamma, \Psi \models \Phi \iff \Gamma \models \Psi \rightarrow \Phi \]

Yes, but only if \( \Psi \) is closed.
Do we have a deduction theorem at hand?

\[ \Gamma, \Psi \models \Phi \iff \Gamma \models \Psi \rightarrow \Phi \]

? 

Yes, but only if \( \Psi \) is closed.

From now on only **closed** formulas are considered.
Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>left side</th>
<th>right side</th>
</tr>
</thead>
</table>

- $t \in \text{Term}_\Sigma$ an arbitrary ground term (no variables)
- $c \text{ new constant}$
### Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>all</th>
<th>left side</th>
<th>right side</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \Gamma, \forall x.\Phi(x), {x/t}\Phi(x) \implies \Delta )</td>
<td>( \Gamma \implies {x/c}\Phi(x), \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, \forall x.\Phi(x) \implies \Delta )</td>
<td>( \Gamma \implies \forall x.\Phi(x), \Delta )</td>
</tr>
</tbody>
</table>

- \( t \in Term_\Sigma \) an arbitrary ground term (no variables)
- \( c \) new constant
Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>left side</th>
<th>right side</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td></td>
</tr>
<tr>
<td>$\Gamma, \forall x. \Phi(x), {x/t} \Phi(x) \implies \Delta$</td>
<td>$\Gamma \implies {x/c} \Phi(x), \Delta$</td>
</tr>
<tr>
<td>$\Gamma, \forall x. \Phi(x) \implies \Delta$</td>
<td>$\Gamma \implies \forall x. \Phi(x), \Delta$</td>
</tr>
<tr>
<td>ex.</td>
<td></td>
</tr>
<tr>
<td>$\Gamma \implies {x/t} \Phi(x), \exists x. \Phi(x), \Delta$</td>
<td>$\Gamma, {x/c} \Phi(x) \implies \Delta$</td>
</tr>
<tr>
<td>$\Gamma \implies \exists x. \Phi(x), \Delta$</td>
<td>$\Gamma, \exists x. \Phi(x) \implies \Delta$</td>
</tr>
</tbody>
</table>

- $t \in \text{Term}_\Sigma$ an arbitrary ground term (no variables)
- $c$ new constant
Sequent Calculus for FOL

<table>
<thead>
<tr>
<th></th>
<th>left side</th>
<th>right side</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>$\Gamma, \forall x.\Phi(x), {x/t}\Phi(x) \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow {x/c}\Phi(x), \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \forall x.\Phi(x) \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \forall x.\Phi(x), \Delta$</td>
</tr>
<tr>
<td>ex.</td>
<td>$\Gamma \Rightarrow {x/t}\Phi(x), \exists x.\Phi(x), \Delta$</td>
<td>$\Gamma, {x/c}\Phi(x) \Rightarrow \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \Rightarrow \exists x.\Phi(x), \Delta$</td>
<td>$\Gamma, \exists x.\Phi(x) \Rightarrow \Delta$</td>
</tr>
<tr>
<td>insert</td>
<td>$\Gamma, x \triangledown y \Rightarrow {x/y}\Phi(x), \Delta$</td>
<td>—</td>
</tr>
<tr>
<td>eq.</td>
<td>$\Gamma, x \triangledown y \Rightarrow \Phi(x), \Delta$</td>
<td>—</td>
</tr>
</tbody>
</table>

- $t \in Term_{\Sigma}$ an arbitrary ground term (no variables)

- $c$ new constant
Explaining the Rules (I)

The following description shall explain the first-order calculus rules on an intuitive (informal) level. For the remaining section all mentioned terms are ground terms, this means they contain no variables.

**all left** If a $\forall x. \Phi(x)$ occurs in the premise, one can add an instantiation with an arbitrary term $t$ to the premises. This is sound as $\{x/t\} \Phi(x)$ holds for all elements of the universe, in particular for the element $t$ is evaluated to. In contrast to the former rules one keeps the quantified formula in the antecedent as one may require more than one instantiation.

**ex. left** $\exists x. \Phi(x)$ can be replaced by $\{x/c\} \Phi$ where $c$ is a new constant. $c$ is thought to be evaluated to the element for which $\Phi(x)$ holds. An already existing term $t$ must not be used as its value is already fixed but in general not to the element satisfying $\Phi(x)$. 
all right  A common way to show that $\forall x. \Phi(x)$ holds, is to take an element of an arbitrary value. In other words, if $\{x/c\} \Phi(x)$ can be shown for a new constant $c$ then the result can be generalised, as no assumptions about the value of $c$ have been made.

In contrast, the generalisation is not possible if an already existing term $t$ is used instead. The value of $t$ has been already fixed to a certain value, which may randomly satisfy $\Phi(x)$, but this may not necessarily be the case for all other elements of the universe (similar to: 2, 3, 5, 7 are primes, so all odd numbers are primes).

ex. right  If $\exists x. \Phi(x)$ has to be proven, one can try to prove it for an arbitrary term $t$. If one uses the wrong term $t$, this means a term for which $\Phi(x)$ is false it is not worse, one only gets false on the right side, which is the neutral element of $\lor$ and so it can just be removed from the sequent. The existential quantified formula is not removed from the sequent, so that one can try to prove the formula for another term $t'$ (sometimes one even has to instantiate the existential quantifiers and all instances are required).
Towards Program Verification

Vertical Verification

- Prove that the implementation fulfills the specification (equivalence for complete specifications)
Towards Program Verification

Vertical Verification

- Prove that the implementation fulfills the specification (equivalence for complete specifications)
- Reasoning about programs
Towards Program Verification

Vertical Verification

- Prove that the implementation fulfills the specification (equivalence for complete specifications)
- Reasoning about programs
- Formalise program properties as formulas of Dynamic Logic
Towards Program Verification

Vertical Verification

- Prove that the implementation fulfills the specification (equivalence for complete specifications)
- Reasoning about programs
- Formalise program properties as formulas of Dynamic Logic

In contrast to testing, verification can show the absence of errors
Do we really need another kind of logics?

»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
State Dependance of Truth Values

What is the truth value of

 ques?

The value of program variable $x$ is 3.' ques

May vary during the execution time of a program.

For example, after the execution of

the value is

the value is

Reasoning about programs must consider the current program state.
What is the truth value of

\[ \text{The value of program variable } x \text{ is } 3. \]

May vary during the execution time of a program.
State Dependance of Truth Values

What is the truth value of

? 'The value of program variable \( x \) is 3.' ?

May vary during the execution time of a program.

For example, after the execution of

- \( x = 3; \ \text{the value is } true \)
State Dependence of Truth Values

What is the truth value of

? 'The value of program variable $x$ is 3.' ?

May vary during the execution time of a program.

For example, after the execution of

- $x=3$; the value is $true$

- $x=4$; the value is $false$
State Dependence of Truth Values

What is the truth value of

? 'The value of program variable $x$ is 3.' ?

May vary during the execution time of a program.

For example, after the execution of

- $x=3$; the value is $true$
- $x=4$; the value is $false$

⇒ Reasoning about programs must consider the current program state.
Dynamic Logics for a simple 'while' language

Signature

$$\Sigma = (\mathcal{P}, \mathcal{F}, \Pi_0, \mathcal{O} \cup \{\cdot, [\cdot]\}), Sorts = \{\text{int}, \text{boolean}\}$$
Dynamic Logics for a simple ’while’ language

**Signature**

\[ \Sigma = (\mathcal{P}, \mathcal{F}, \Pi_0, \mathcal{O} \cup \{\cdot, [.]\}), \text{Sorts} = \{\text{int, boolean}\} \]

\( \Pi_0 \) is a set of atomic programs (e.g. \( \alpha, \beta \))
Dynamic Logics for a simple 'while' language

Signature

$\Sigma = (\mathcal{P}, \mathcal{F}, \Pi_0, \mathcal{O} \cup \{\langle \cdot, [\cdot] \})$, $Sorts = \{\text{int}, \text{boolean}\}$

$\Pi_0$ is a set of atomic programs (e.g. $\alpha, \beta$)

Definition of Programs $\Pi$
Dynamic Logics for a simple 'while' language

Signature

\[ \Sigma = (\mathcal{P}, \mathcal{F}, \Pi_0, \mathcal{O} \cup \{\cdot, [\cdot]\}), \text{Sorts} = \{\text{int, boolean}\} \]

\(\Pi_0\) is a set of atomic programs (e.g. \(\alpha, \beta\))

Definition of Programs \(\Pi\)

If \(\alpha, \beta \in \Pi_0\) and \(b\) a term of sort \(\text{bool}\) then

- \(\alpha; \beta\)
Dynamic Logics for a simple 'while' language

Signature

\( \Sigma = (\mathcal{P}, \mathcal{F}, \Pi_0, \mathcal{O} \cup \{\cdot, [\cdot]\}), Sorts = \{\text{int, boolean}\} \)

\( \Pi_0 \) is a set of atomic programs (e.g. \( \alpha, \beta \))

Definition of Programs \( \Pi \)

If \( \alpha, \beta \in \Pi_0 \) and \( b \) a term of sort \( \text{bool} \) then

- \( \alpha; \beta \)

- if \((b)\) then \(\{\alpha\}\) else \(\{\beta\}\)
Dynamic Logics for a simple 'while' language

Signature

\[ \Sigma = (\mathcal{P}, \mathcal{F}, \Pi_0, \mathcal{O} \cup \{\langle \cdot \rangle, [.]\}), \text{Sorts} = \{\text{int}, \text{boolean}\} \]

\(\Pi_0\) is a set of atomic programs (e.g. \(\alpha, \beta\))

Definition of Programs \(\Pi\)

If \(\alpha, \beta \in \Pi_0\) and \(b\) a term of sort \(\text{bool}\) then

- \(\alpha; \beta\)
- if \((b)\) then \(\{\alpha\}\) else \(\{\beta\}\)
- while \((b)\{\alpha\}\)

are programs in \(\Pi\).
Definition of Terms

Defined as in first-order logics. But we distinct between

- **rigid** terms, which are meant to be state independant
Definition of Terms

Defined as in first-order logics. But we distinct between

- **rigid** terms, which are meant to be state independant

- **non-rigid** (or flexible) terms, whose value (interpretation) will depend on the current program state
Terms and Formulas of Dynamic Logics

Definition of Terms

Defined as in first-order logics. But we distinct between

- **rigid** terms, which are meant to be state independant
- **non-rigid** (or flexible) terms, whose value (interpretation) will depend on the current program state

Definition of Formulas

All formulas of FOL are also dynamic logic formulas (DL formulas).
Terms and Formulas of Dynamic Logics

Definition of Terms

Defined as in first-order logics. But we distinct between

- **rigid terms**, which are meant to be state independant

- **non-rigid** (or flexible) terms, whose value (interpretation) will depend on the current program state

Definition of Formulas

All formulas of FOL are also dynamic logic formulas (DL formulas).

If $\alpha$ is a program and $\Phi$ a formula then
Terms and Formulas of Dynamic Logics

**Definition of Terms**

Defined as in first-order logics. But we distinguish between

- **rigid** terms, which are meant to be state independent

- **non-rigid** (or flexible) terms, whose value (interpretation) will depend on the current program state

**Definition of Formulas**

All formulas of FOL are also dynamic logic formulas (DL formulas).

If $\alpha$ is a program and $\Phi$ a formula then

$\langle \alpha \rangle \Phi$ is a DL-Formula
Terms and Formulas of Dynamic Logics

Definition of Terms

Defined as in first-order logics. But we distinct between

- **rigid** terms, which are meant to be state independant
- **non-rigid** (or flexible) terms, whose value (interpretation) will depend on the current program state

Definition of Formulas

All formulas of FOL are also dynamic logic formulas (DL formulas).

If $\alpha$ is a program and $\Phi$ a formula then

$\langle \alpha \rangle \Phi$ is a DL-Formula

$[\alpha] \Phi$ is a DL-Formula
**Semantics of Dynamic Logic - Kripke Structure**

**Kripke-Structure** \( \mathcal{K} = (\text{States}, \rho) \)

**where** \( s \in \text{State}, s = (\mathcal{U}, I) \) and \( \rho : \Pi_0 \rightarrow \text{States} \times \text{States} \)
Kripke-Structure $\mathcal{K} = (\text{States}, \rho)$

where $s \in \text{State}, s = (\mathcal{U}, I)$ and $\rho : \Pi_0 \rightarrow \text{States} \times \text{States}$

$\rho(\alpha)$
Kripke-Structure $\mathcal{K} = (\text{States}, \rho)$

where $s \in \text{State}, s = (U, I)$ and $\rho : \Pi_0 \rightarrow \text{States} \times \text{States}$

$\rho(\alpha), \rho(\beta)$
There exists an \( \alpha \)-reachable state, such that \( \Phi \) holds.
There exists an $\alpha$-reachable state, such that $\Phi$ holds.

$[\alpha] \Phi$  $\Phi$ holds in all $\alpha$-reachable states.
There exists an $\alpha$-reachable state, such that $\Phi$ holds.

$\Phi$ holds in all $\alpha$-reachable states.

What does this mean in terms of program execution?
There exists an $\alpha$-reachable state, such that $\Phi$ holds.

$\Phi$ holds in all $\alpha$-reachable states.

What does this mean in terms of program execution?

$\langle \cdot \rangle$: total correctness; $[\cdot]$: partial correctness
There exists an $\alpha$-reachable state, such that $\Phi$ holds.

$[\alpha] \Phi$ $\Phi$ holds in all $\alpha$-reachable states.

What does this mean in terms of program execution?

$\langle \cdot \rangle$: total correctness; $[\cdot]$: partial correctness

**Duality:** $\langle \alpha \rangle \Phi$ iff. $\neg [\alpha] \neg \Phi$
Semantics of Dynamic Logic

Let $\mathcal{P} = \{A, B, C\}$, $\mathcal{D} = \mathbb{N}$ and

$s_1 : I = \{A, B\}$, $s_2 : I = \{C\}$, $s_4 : I = \{A\}$
Semantics of Dynamic Logic

Let \( \mathcal{P} = \{A, B, C\} \), \( \mathcal{D} = \mathbb{IN} \) and

\[ s_1 : I = \{A, B\}, \quad s_2 : I = \{C\}, \quad s_4 : I = \{A\} \]

\[ s_1 \models (\alpha) A? \]
Semantics of Dynamic Logic

Let $\mathcal{P} = \{A, B, C\}$, $\mathcal{D} = \mathbb{IN}$ and

$s_1 : I = \{A, B\}$, $s_2 : I = \{C\}$, $s_4 : I = \{A\}$

$s_1 \models \langle \alpha \rangle A$ (ok),
Let $\mathcal{P} = \{A, B, C\}$, $\mathcal{D} = \mathbb{IN}$ and

$s_1 : I = \{A, B\}$, $s_2 : I = \{C\}$, $s_4 : I = \{A\}$

$s_1 \models \langle \alpha \rangle A \ (\text{ok}), \ s_1 \models \langle \beta \rangle A$?
Semantics of Dynamic Logic

Let $\mathcal{P} = \{A, B, C\}$, $\mathcal{D} = \mathbb{IN}$ and

$s_1: I = \{A, B\}$, $s_2: I = \{C\}$, $s_4: I = \{A\}$

$s_1 \models \langle \alpha \rangle A$ (ok), $s_1 \models \langle \beta \rangle A$ (—)
Semantics of Dynamic Logic

Let $\mathcal{P} = \{A, B, C\}$, $\mathcal{D} = \mathbb{IN}$ and

$s1 : I = \{A, B\}$, $s2 : I = \{C\}$, $s4 : I = \{A\}$

$s1 \models \langle \alpha \rangle A$ (ok), $s1 \models \langle \beta \rangle A$ (→)

$s5 \models \langle \beta \rangle A$?
Let $\mathcal{P} = \{A, B, C\}$, $\mathcal{D} = \mathbb{IN}$ and

$s_1 : I = \{A, B\}$, $s_2 : I = \{C\}$, $s_4 : I = \{A\}$

$s_1 \models \langle \alpha \rangle A$ (ok), $s_1 \models \langle \beta \rangle A$ (–)

$s_5 \models \langle \beta \rangle A$ (–),
Semantics of Dynamic Logic

Let $\mathcal{P} = \{A, B, C\}$, $\mathcal{D} = \mathbb{IN}$ and

$s_1 : I = \{A, B\}$, $s_2 : I = \{C\}$, $s_4 : I = \{A\}$

$s_1 \models \langle \alpha \rangle A$ (ok), $s_1 \models \langle \beta \rangle A$ (—)

$s_5 \models \langle \beta \rangle A$ (—), $s_5 \models [\beta] A$?
Semantics of Dynamic Logic

Let $\mathcal{P} = \{A, B, C\}$, $\mathcal{D} = \mathbb{IN}$ and

$s_1 : I = \{A, B\}, s_2 : I = \{C\}, s_4 : I = \{A\}$

$s_1 \models \langle \alpha \rangle A \text{ (ok)}, \quad s_1 \models \langle \beta \rangle A \text{ (—)}$

$s_5 \models \langle \beta \rangle A \text{ (—)}, \quad s_5 \models [\beta] A \text{ (ok)}$
The atomic programs are assignments:

\[ x = t \quad (\text{sort}(x) = \text{sort}(t) = \text{int}) \]
A 'While'-Language with Assignments (I)

- The atomic programs are assignments:
  \[ x = t \quad (sort(x) = sort(t) = int) \]

- Terms are arithmetical expressions (functions \(+,-,\ast\) )
A ’While’-Language with Assignments (I)

- The atomic programs are assignments:
  
  \[ x = t \quad (\text{sort}(x) = \text{sort}(t) = \text{int}) \]

- Terms are arithmetical expressions (functions \( +, -, \times \))

- Conditions are built with \( > \) and \( \geq \)
A 'While'-Language with Assignments (I)

- The atomic programs are assignments:
  \[ x = t \quad (\text{sort}(x) = \text{sort}(t) = \text{int}) \]

- Terms are arithmetical expressions (functions +, −, *)

- Conditions are built with > and >=

Example

```plaintext
y=1;
x=3;
while (x>0) {
    y=y*x;
    x=x-1;
}
```
A ’While’-Language with Assignments(II)

**States** $s = (U, I, \sigma)$

- have all the same universe $U$
States $s = (U, I, \sigma)$

- have all the same universe $U$
- predicate symbols are rigid
States \( s = (U, I, \sigma) \)

- have all the same universe \( U \)
- predicate symbols are rigid

Further agreement:

- Logic variables vs. program variables:
  
  Program variables cannot be quantified. Their value depends on the current state. Therefore each state contains a function

  \[ \sigma : \text{ProgVar} \rightarrow U. \]
A 'While'-Language with Assignments(II)

States $s = (U, I, \sigma)$

- have all the same universe $U$
- predicate symbols are rigid

Further agreement:

- Logic variables vs. program variables:
  Program variables cannot be quantified. Their value depends on the current state. Therefore each state contains a function $\sigma : \text{ProgVar} \rightarrow U$.
  On the other hand, logic variables are not allowed to occur in programs and they must be bound by a quantifier.
Local Validity

There is some choice selecting the consequence relation $\models$.

The deduction theorem holds for the local version:

$$\Gamma \models \Phi$$

iff.

for all states $g$: if $g \models \Gamma$ then $g \models \Phi$
Local Validity

There is some choice selecting the consequence relation $\models$.

The deduction theorem holds for the local version:

$$\Gamma \models \Phi$$

iff.

for all states $g$: if $g \models \Gamma$ then $g \models \Phi$

(global version:

$$\Gamma \models \Phi$$

iff.

for all states $g$: $g \models \Gamma$ then for all states $g$: $g \models \Phi$)
Sequent Calculus Rules

\[\Gamma, b \vdash true \implies \langle \alpha \rangle \Phi, \Delta \quad \Gamma \implies b \vdash true, \langle \beta \rangle \Phi, \Delta\]

\[\Gamma \implies \langle \text{if } (b) \text{ then } \alpha; \text{ else } \beta; \rangle \Phi, \Delta\]
Sequent Calculus Rules

**IF-ELSE**

\[
\frac{\Gamma, b \vdash \text{true} \implies \langle \alpha \rangle \Phi, \Delta}{\Gamma \implies \langle \text{if } (b) \text{ then } \alpha; \text{ else } \beta; \rangle \Phi, \Delta}
\]

**Assignment**

\[
\frac{\Gamma^{x \leftarrow y}, x \vdash t \vdash \Phi, \Delta^{x \leftarrow y}}{\Gamma \vdash \langle x = t \rangle \Phi, \Delta (y \text{ new variable})}
\]
Sequent Calculus Rules

IF-ELSE

\[ \Gamma, b \vdash \text{true} \implies \langle \alpha \rangle \Phi, \Delta \quad \Gamma \implies b \vdash \text{true}, \langle \beta \rangle \Phi, \Delta \]

\[ \Gamma \implies \langle \text{if} \ (b) \ \text{then} \ \alpha; \ \text{else} \ \beta; \rangle \Phi, \Delta \]

Assignment

\[ \Gamma^{x \leftarrow y}, x \vdash t \vdash \Phi, \Delta^{x \leftarrow y} \]

\[ \Gamma \vdash \langle x = t \rangle \Phi, \Delta \]

(y new variable)
Example

DEMO