Contents

- Overview of KeY
- UML and its semantics
- Introduction to OCL
- **Specifying requirements with OCL**
- Modelling of Systems with Formal Semantics
- Propositional & First-order logic, sequent calculus
- OCL to Logic, horizontal proof obligations, using KeY
- Dynamic logic, proving program correctness
- Java Card DL
- Vertical proof obligations, using KeY
- Wrap-up, trends
Reminder: Object Diagrams and OCL

id0815:Person
name = "Jane"
age = 5

harley17:Bike
colour = idBlack

idBlack:Colour
black() = idBlack
white() = idWhite
red() = idRed

id0825:Person
name = "Paul"
age = 25

bmw3:Car
colour = idWhite

idWhite:Colour
black() = idBlack
white() = idWhite
red() = idRed

idRed:Colour
black() = idBlack
white() = idWhite
red() = idRed

context Vehicle
inv: self.owner.age >= 18

Reminder: Object Diagrams and OCL

```
context Vehicle
inv: self.owner.age >= 18

I(Vehicle) = {harley17, bmw3}
```
Reminder: Object Diagrams and OCL

**context** Vehicle
**inv:** self.owner.age $\geq 18$ ✓

$I(Vehicle) = \{harley17, bmw3\}$

$\Rightarrow harley17.I(owner).I(age) \geq 18 \text{ and } bmw3.I(owner).I(age) \geq 18$
Reminder: Object Diagrams and OCL

context Vehicle
inv: self.owner.age ≥ 18

$I(Vehicle) = \{\text{harley17, bmw3}\}$

$\Rightarrow \text{harley17}.I(\text{owner}).I(\text{age}) \geq 18 \text{ and } \text{bmw3}.I(\text{owner}).I(\text{age}) \geq 18$

$I(\text{owner}) : \text{Vehicle} \rightarrow \text{Person}$
Reminder: Object Diagrams and OCL

context Vehicle
inv: self.owner.age >= 18

\( I(Vehicle) = \{ harley17, bmw3 \} \)

\( \Rightarrow harley17.I(owner).I(age) \geq 18 \text{ and } bmw3.I(owner).I(age) \geq 18 \)

\( I(owner)(harley17) = I(owner)(bmw3) = id0825 \)
Reminder: Object Diagrams and OCL

context Vehicle
inv: self.owner.age >= 18

\( I(Vehicle) = \{\text{harley17, bmw3}\} \)
\( \Rightarrow \text{harley17}.I(owner).I(age) \geq 18 \text{ and bmw3}.I(owner).I(age) \geq 18 \)
\( I(owner)(\text{harley17}) = I(owner)(\text{bmw3}) = \text{id0825} \)
\( \Rightarrow \text{id0825}.I(age) \geq 18 \)
context Vehicle
inv: self.owner.age >= 18  

$I(Vehicle) = \{ \text{harley17, bmw3} \}$

$⇒ \text{harley17}.I(\text{owner}).I(\text{age}) \geq 18$ and $\text{bmw3}.I(\text{owner}).I(\text{age}) \geq 18$

$I(\text{owner})(\text{harley17}) = I(\text{owner})(\text{bmw3}) = \text{id0825}$

$⇒ \text{id0825}.I(\text{age}) \geq 18$

$I(\text{age})(\text{id0825}) = 25 \Rightarrow 25 \geq 18$  

OCL and Formal Proofs

Snapshots provide formal semantics for UML and OCL

⇒ can formally prove properties of model and implementation
OCL and Formal Proofs

Snapshots provide formal semantics for UML and OCL

⇒ can formally prove properties of model and implementation

Examples:

- Invariant of class $A$ implies invariant of class $B$
  
  For each snapshot $I$: if $A$’s invariant holds in $I$, then so does $B$’s

  Horizontal verification problem (within specification)

- Implementation of operation $m$ fulfills its contract
  
  For each snapshot $I$: if precondition of $m$ holds in $I$, then its postcondition holds in snapshot $I'$ produced by execution of $m$

  Vertical verification problem (implementation against specification)
Snapshots and States: Static View

Snapshots have **static** and **dynamic** part.
Snapshots and States: Static View

Snapshots have **static** and **dynamic** part

**Static** (object diagram): objects, attribute values, associations

Static part of snapshot similar to execution **state** of program

Denote such states with $s$, set of all states $S$ (infinite!)

Think of one single state as **object diagram**

Proving horizontal verification problem: **state inclusion**
Snapshots and States: Static View

Snapshots have **static** and **dynamic** part

**Static** (object diagram): objects, attribute values, associations

Static part of snapshot similar to execution **state** of program

Denote such states with \( s \), set of all states \( S \) (infinite!)

Think of one single state as **object diagram**

Proving horizontal verification problem: **state inclusion**

**Example:** Let \( \text{inv}_A \) be invariant of class \( A \), \( \text{inv}_B \) invariant of class \( B \)

\[
\{ s \in S \mid \text{inv}_A \text{ holds in } s \} \subseteq \{ s \in S \mid \text{inv}_B \text{ holds in } s \}
\]
Snapshots and States: Dynamic View

Program state \( s = \) static part of snapshot
Set of all states \( S \)

**Dynamic** part of snapshot:

Semantics of operations \( m: \rho(m): S \rightarrow S \)

Operation can be seen as **state transformer**

For each \( m \) and \( s \in S \), result state \( \rho(m)(s) \)

\( \rho \) is **partial function**: programs deterministic, may not terminate

Proving vertical verification problem: **state reachability**
Snapshots and States: Dynamic View

Program state \( s = \) static part of snapshot
Set of all states \( S \)

**Dynamic** part of snapshot:

Semantics of operations \( m: \rho(m): S \to S \)

Operation can be seen as **state transformer**

For each \( m \) and \( s \in S \) **result state** \( \rho(m)(s) \)

\( \rho \) is **partial function**: programs deterministic, may not **terminate**

Proving vertical verification problem: **state reachability**

Example: Let \( \text{pre} \) be precondition, \( \text{post} \) postcondition of \( m \)

Does \( \text{post} \) hold in all states \( s' \in \{ \rho(m)(s) \mid s \text{ satisfies } \text{pre} \} \)?

Does \( \text{post} \) hold in all states \( s' \) **that can be reached via** \( m \)

from any state \( s \) satisfying \( \text{pre} \)?
(Deterministic) Labelled Transition System (LTS) $K = (S, \rho)$:

$S$ set of states, $\rho: \text{Method} \rightarrow (S \rightarrow S)$ (takes a program and returns a map from $S$ to $S$), $\alpha = \rho(m)$, $\beta = \rho(m')$

Infinite number of states $\Rightarrow$ need theorem proving (or approximation)
Dynamic Part of Snapshots as LTS

- Each state is a static snapshot (i.e., object diagram) with the current objects and values.

- If $\rho(m)$ takes, say, state $s_1$ into $s_4$, then a directed edge from $s_1$ to $s_4$ labelled with $\rho(m)$ is present in $K$.

- $\rho(m)$ is then a (possibly infinite) number of pre-/post execution state pairs.

- There is no explicit notion of initial state.

- One may consider as initial states those that satisfy the precondition of a distinguished `main` method (and possibly the invariant of its class).
Encoding Verification Problem in Logic

UML Model

CASE Tool

Java (Card)
(partial implementation)

Java (Card)

CASPAR
Univ Karlsruhe

AST

OCL
(partial specification)

OCL

OCL Parser
Univ Dresden

Translation
formula synthesis (vert. verif.)

FOL/Java Card DL

Interactive/Automated Theorem Prover

Formal proof
Why translate OCL into Logic?
Why translate OCL into Logic?

**Difficult and expensive to develop theorem prover for a formalism**

- OCL only one of many specification languages (JML, RSL, etc.)
- OCL prone to change (1.3, 1.4, 1.5, . . . , 2.0, . . . ?)
- First order logic (FOL) well understood, mature tools
  “FOL in verification like the Reals in Calculus”
Why translate OCL into Logic?

Difficult and expensive to develop theorem prover for a formalism

- OCL only one of many specification languages (JML, RSL, etc.)
- OCL prone to change (1.3, 1.4, 1.5, ..., 2.0, ... ?)
- First order logic (FOL) well understood, mature tools
  “FOL in verification like the Reals in Calculus”

OCL not designed for verification, programming language independent

- OCL (UML) doesn’t know about implementation of operations
  Need to incorporate Java data types and programs
- OCL not designed to express verification problems
- OCL doesn’t know about (class) initialization (<2.0)
Contents

- Overview of KeY
- UML and its semantics
- Introduction to OCL
- Specifying requirements with OCL
- Modelling of Systems with Formal Semantics
- Propositional & First-order logic, sequent calculus
- OCL to Logic, horizontal proof obligations, using KeY
- Dynamic logic, proving program correctness
- Java Card DL
- Vertical proof obligations, using KeY
- Wrap-up, trends
Formalisation

Real World → Formalisation → Formal Model
Formalisation

Real World

Formal Language

Formal Semantics
Formalisation

Real World

UML
OCL
Java

$I, \rho$

Snapshots/LTS
Formalisation

Real World

Snapshots/LTS
“infinite”

UML
OCL
Java

I, \rho
Formalisation

Real World

UML
OCL
Java

Snapshots/LTS
“infinite”

Calculus
“finite”
Formal Verification

Real World

UML
OCL
Java

Snapshot/
LTS
“infinite”

Calculus
“finite”

I, ρ

Formal Verification

Real World

UML OCL Java

I, ρ

Obj. Diagr. Snapshot/ LTS

Calculus

Formal Verification

Real World

UML
OCL
Java

Obj. Diagr.
Snapshot/
LTS

Calculus

Translation

I, ρ

Translation

I, ρ

FO Logic
Dyn. Logic

FO Interp.
Kripke Str.
Formal Verification

Real World

UML

OCL

Java

Obj. Diagr.

Snapshot/
LTS

Translation

I, ρ

I, ρ

FO Logic

Dyn. Logic

FO Interp.

Kripke Str.

Sequent
Calculus

I, ρ, |=

Translation
Syntax, Semantics, Calculus

Syntax
"Formula"

Semantics
"Valid"

Calculus
"Derivable"
Syntax, Semantics, Calculus

Syntax
“Formula”

Semantics
“Valid”

Calculus
“Derivable”

Completeness

$I, \rho, \vdash$
Syntax, Semantics, Calculus

Syntax
“Formula”

Semantics
“Valid”

Calculus
“Derivable”

Completeness

Soundness

$\Gamma, \rho, \models$

$\vdash$

Propositional Logic

Propositional Formulas

$I, \models$

Mapping Variables into \{true, false\}

$\vdash$

Sequent Calculus SAT solver

Propositional Logic

Propositional Formulas

Mapping Variables into \{true, false\}

I, \models

Sequent Calculus
Contents

- Overview of KeY
- UML and its semantics
- Introduction to OCL
- Specifying requirements with OCL
- Modelling of Systems with Formal Semantics
- Propositional & First-order logic, sequent calculus
- OCL to Logic, horizontal proof obligations, using KeY
- Dynamic logic, proving program correctness
- Java Card DL
- Vertical proof obligations, using KeY
- Wrap-up, trends
Propositional Logic: Syntax

- Propositional Formulas
- Mapping Variables into \{true, false\}
- Sequent Calculus

I, \models
Syntax of Propositional Logic

The **Signature:**

Propositional Variables \( \mathcal{P} = \{ p_i \mid i \in \mathbb{N} \} \) with type **Boolean**
Syntax of Propositional Logic

The **Signature:**

**Propositional Variables** $\mathcal{P} = \{p_i | i \in \mathbb{N}\}$ with **type** Boolean

**Connectives** $\{\text{true, false, } \&, |, !, \rightarrow, \leftrightarrow\}$
Syntax of Propositional Logic

The **Signature:**

Propositional Variables $\mathcal{P} = \{p_i | i \in \mathbb{N}\}$ with type **Boolean**

**Connectives** \{true, false, &, |, !, ->, <->\}

**Propositional Formulas** $F_0$ (all have type **Boolean**)

Truth constants ‘true’, ‘false’ and variables $\mathcal{P}$ are formulas
Syntax of Propositional Logic

- **The Signature:**
  
  Propositional Variables $\mathcal{P} = \{ p_i \mid i \in \mathbb{N} \}$ with type Boolean

- **Connectives** $\{ \text{true, false, } \&, \ |, \ !, \ -\rightarrow, \ <\rightarrow \}$

**Propositional Formulas** $For_0$ (all have type Boolean)

- Truth constants ‘true’, ‘false’ and variables $\mathcal{P}$ are formulas

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

  $!G, \ (G \& H), \ (G | H), \ (G \rightarrow H), \ (G \leftrightarrow H)$

  are also formulas

- There are no other formulas (inductive definition)
Propositional Logic: Semantics

- Propositional Formulas
  - Mapping Variables into \{true, false\}
  - Sequent Calculus

$I, \models$
Semantics of Propositional Logic

**Interpretation** $\mathcal{I}$

Assigns a truth value to each propositional variable

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

**Interpretation** \( \mathcal{I} \)

Assigns a truth value to each propositional variable

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

**Valuation function** \( val_\mathcal{I} : \text{extension of } \mathcal{I} \text{ to } \text{For}_0 \)

\[
val_\mathcal{I} : \text{For}_0 \rightarrow \{true, false\}
\]
Interpretation $\mathcal{I}$

Assigns a truth value to each propositional variable

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

Valuation function $\text{val}_\mathcal{I}$: extension of $\mathcal{I}$ to $\text{For}_0$

$$\text{val}_\mathcal{I} : \text{For}_0 \rightarrow \{true, false\}$$

$$\text{val}_\mathcal{I}(p_i) = \mathcal{I}(p_i)$$

$$\text{val}_\mathcal{I}(\text{true}) = true$$

$$\text{val}_\mathcal{I}(\text{false}) = false$$

$$\text{val}_\mathcal{I}(G \rightarrow H) = \begin{cases} true & \text{if } \text{val}_\mathcal{I}(G) = false \text{ or } \\
false & \text{otherwise} \end{cases}$$

$\mathcal{I}$ satisfies $G$ if $\text{val}_\mathcal{I}(G) = true$; otherwise, it falsifies $G$. 
Example

Formula

\[ p \rightarrow (q \rightarrow p) \]
Example

Formula

\[ p \rightarrow (q \rightarrow p) \]

Interpretation (one of four that are possible)

\[ I(p) = true \]
\[ I(q) = false \]
Example

Formula

\[ p \rightarrow (q \rightarrow p) \]

Interpretation (one of four that are possible)

\[ \mathcal{I}(p) = true \]
\[ \mathcal{I}(q) = false \]

Valuation

\[ val_{\mathcal{I}}(q \rightarrow p) = true \]
Example

Formula

\[ p \rightarrow (q \rightarrow p) \]

Interpretation (one of four that are possible)

\[ I(p) = true \]
\[ I(q) = false \]

Valuation

\[ val_I(q \rightarrow p) = true \]
\[ val_I(p \rightarrow (q \rightarrow p)) = true \]
Semantic Notions

Let $G \in For_0$, $\Gamma \subseteq For_0$

- **Validity Relation** $\models$
  
  $G$ is valid in $\mathcal{I}$ iff $val_{\mathcal{I}}(G) = true$ (write: $\mathcal{I} \models G$)

  A formula that is valid in some interpretation is **satisfiable**

- $\Gamma$ entails $G$ ($\Gamma \models G$) iff for all interpretations $\mathcal{I}$:
  
  If $\mathcal{I} \models H$ for all $H \in \Gamma$ then also $\mathcal{I} \models G$

- If $G$ is valid in any interpretation, i.e
  
  $\emptyset \models G$ (short: $\models G$)

  then $G$ is called **logically valid**
Propositional Logic Examples

\[ p \& ((\neg p) \lor q) \]

Satisfiable?
Propositional Logic Examples

$p \& ((\neg p) \mid q)$

Satisfiable? Yes
Propositional Logic Examples

\[ p \land ((\neg p) \lor q) \]

Satisfiable? Yes

Satisfying Interpretation?
Propositional Logic Examples

$p \& ((\neg p) \mid q)$

Satisfiable? Yes

Satisfying Interpretation? $I(p) = true, I(q) = true$
Propositional Logic Examples

\[ p \& ((\lnot p) \lor q) \]

**Satisfiable?**  Yes

**Satisfying Interpretation?** \( \mathcal{I}(p) = true, \mathcal{I}(q) = true \)

\[ p \& ((\lnot p) \lor q) \models q \lor r \]

**Does this hold?**
Propositional Logic Examples

\[ p \& ((\neg p) \mid q) \]

**Satisfiable?**  Yes

**Satisfying Interpretation?**  \( I(p) = \text{true}, I(q) = \text{true} \)

\[ p \& ((\neg p) \mid q) \models q \mid r \]

**Does this hold?**  Yes.  **Why?**
Propositional Logic

Propositional Formulas

\[ \models \]

Mapping Variables into \{true, false\}

\[ \vdash \]

Sequent Calculus
Reasoning by Syntactic Transformation

Establish $G$ by finite syntactic transformations of $G$
Reasoning by Syntactic Transformation

Establish $|$ $G$ by finite syntactic transformations of $G$

(Logic) Calculus: a set of syntactic transformation rules $\mathcal{R}$ defining a property $\vdash$ over $For_0$ such that $|$ $G$ iff $\vdash G$ (G is derivable)

$|$ $G$ implies $\vdash G$ (Completeness) $\vdash G$ implies $|$ $G$ (Soundness)
Reasoning by Syntactic Transformation

Establish $\models G$ by finite syntactic transformations of $G$

(Logic) Calculus: a set of syntactic transformation rules $\mathcal{R}$ defining a property $\vdash$ over $\text{For}_0$ such that $\models G$ iff $\vdash G$ ($G$ is derivable)

$\models G$ implies $\vdash G$ (Completeness) $\vdash G$ implies $\models G$ (Soundness)

Sequent Calculus based on notion of sequent

$$\begin{array}{c}
\underbrace{\psi_1, \ldots, \psi_m} \\
\text{Antecedent}
\end{array} \implies
\begin{array}{c}
\underbrace{\phi_1, \ldots, \phi_n} \\
\text{Succedent}
\end{array}$$

has same semantics as

$$(\psi_1 \& \cdots \& \psi_m) \to (\phi_1 | \cdots | \phi_n)$$

$$\{\psi_1, \ldots, \psi_m\} \models \phi_1 | \cdots | \phi_n$$
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 schema variables \( \Gamma, \phi, \ldots \) that match (sets of) formulas
Characterize infinitely many formulas with a single sequent

\[ \Gamma \implies \Delta, \phi \& \psi \]

Matches any sequent with occurrence of conjunction in succedent

Call \( \phi \& \psi \) main formula and \( \Gamma, \Delta \) side formulas of sequent

Any sequent of the form \( \Gamma, \phi \implies \Delta, \phi \) is logically valid, and is called an axiom
Sequent Calculus Rules

Basic idea: write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible

\[
\text{RULE NAME} \quad \frac{\Gamma_1 == \Delta_1 \cdots \Gamma_r == \Delta_r}{\Gamma == \Delta}
\]

Premisses

Conclusion
Sequent Calculus Rules

**Basic idea:** write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible

**RULE NAME**

\[
\begin{array}{c}
\Gamma_1 ==> \Delta_1 \quad \cdots \quad \Gamma_r ==> \Delta_r \\
\hline
\Gamma ==> \Delta
\end{array}
\]

**Premisses**

**Conclusion**

**Example**

**AND_RIGHT**

\[
\begin{array}{c}
\Gamma ==> \phi, \Delta \\
\Gamma ==> \psi, \Delta \\
\hline
\Gamma ==> \phi \& \psi, \Delta
\end{array}
\]
Sequent Calculus Rules

**Basic idea:** write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible

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

**Example**

**AND_RIGHT**

\[
\begin{array}{c}
\Gamma \implies \phi, \Delta \\
\Gamma \implies \psi, \Delta \\
\hline
\Gamma \implies \phi \& \psi, \Delta
\end{array}
\]

Rules can have zero premisses (iff conclusion is valid, eg. an axiom)
Sequent Calculus Rules

**Basic idea:** write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible

![Rule Diagram]

**Example**

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Premisses</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND_RIGHT</strong></td>
<td>( \Gamma \implies \phi, \Delta )</td>
<td>( \Gamma \implies \phi \land \psi, \Delta )</td>
</tr>
</tbody>
</table>

A rule is **sound** if every interpretation that satisfies each premiss of the rule also satisfies its conclusion (essential property)
Sequent Calculus Rules

**Basic idea:** write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible

**Rule Name**

\[
\begin{array}{c}
\Gamma_1 ==> \Delta_1 \quad \cdots \quad \Gamma_r ==> \Delta_r \\
\Gamma ==> \Delta \\
\end{array}
\]

**Premisses**

**Conclusion**

**Example**

\[
\begin{array}{c}
\Gamma ==> \phi, \Delta \\
\Gamma ==> \psi, \Delta \\
\Gamma ==> \phi \& \psi, \Delta \\
\end{array}
\]

- A rule is **sound** if every interpretation that satisfies each premiss of the rule also satisfies its conclusion (essential property)

- A rule is **complete** if every interpretation that satisfies its conclusion also satisfies each of its premisses (desirable property)
### Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (work on antecedent)</th>
<th>right side (work on succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>[ \Gamma \implies \phi, \Delta ] [ \Gamma, \neg \phi \implies \Delta ]</td>
<td>[ \Gamma, \phi \implies \Delta ] [ \Gamma \implies \neg \phi, \Delta ]</td>
</tr>
</tbody>
</table>
## Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (work on antecedent)</th>
<th>right side (work on succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>$\Gamma \implies \phi, \Delta$</td>
<td>$\Gamma, \phi \implies \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \neg \phi \implies \Delta$</td>
<td>$\Gamma \implies \neg \phi, \Delta$</td>
</tr>
<tr>
<td>and</td>
<td>$\Gamma, \phi, \psi \implies \Delta$</td>
<td>$\Gamma \implies \phi, \Delta$ $\Gamma \implies \psi, \Delta$ $\Gamma \implies \phi &amp; \psi, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \phi &amp; \psi \implies \Delta$</td>
<td></td>
</tr>
</tbody>
</table>

22c181: Formal Methods in Software Engineering – p.28/32
### Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (work on antecedent)</th>
<th>right side (work on succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>not</strong></td>
<td>( \Gamma \implies \phi, \Delta )</td>
<td>( \Gamma, \phi \implies \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, !\phi \implies \Delta )</td>
<td>( \Gamma \implies !\phi, \Delta )</td>
</tr>
<tr>
<td><strong>and</strong></td>
<td>( \Gamma, \phi, \psi \implies \Delta )</td>
<td>( \Gamma \implies \phi, \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, \phi \land \psi \implies \Delta )</td>
<td>( \Gamma \implies \psi, \Delta )</td>
</tr>
<tr>
<td><strong>or</strong></td>
<td>( \Gamma \implies \phi, \Delta ) ( \Gamma, \psi \implies \Delta )</td>
<td>( \Gamma \implies \phi, \psi, \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, \phi \mid \psi \implies \Delta )</td>
<td>( \Gamma \implies \phi \mid \psi, \Delta )</td>
</tr>
</tbody>
</table>
### Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (work on antecedent)</th>
<th>right side (work on succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>(\Gamma \implies \phi, \Delta) (\Gamma, !\phi \implies \Delta)</td>
<td>(\Gamma, \phi \implies \Delta) (\Gamma \implies !\phi, \Delta)</td>
</tr>
<tr>
<td>and</td>
<td>(\Gamma, \phi, \psi \implies \Delta) (\Gamma, \phi &amp; \psi \implies \Delta)</td>
<td>(\Gamma \implies \phi, \Delta) (\Gamma \implies \psi, \Delta) (\Gamma \implies \phi &amp; \psi, \Delta)</td>
</tr>
<tr>
<td>or</td>
<td>(\Gamma, \phi \implies \Delta) (\Gamma, \psi \implies \Delta) (\Gamma, \phi \mid \psi \implies \Delta)</td>
<td>(\Gamma \implies \phi, \psi, \Delta) (\Gamma \implies \phi \mid \psi, \Delta)</td>
</tr>
<tr>
<td>imp</td>
<td>(\Gamma \implies \phi, \Delta) (\Gamma, \psi \implies \Delta) (\Gamma, \phi \implies \psi \implies \Delta)</td>
<td>(\Gamma, \phi \implies \psi, \Delta) (\Gamma \implies \phi \implies \psi, \Delta)</td>
</tr>
</tbody>
</table>
# Rules of Propositional Sequent Calculus

<table>
<thead>
<tr>
<th>main</th>
<th>left side (work on antecedent)</th>
<th>right side (work on succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>( \Gamma \implies \phi, \Delta ) [ smash ] ( \Gamma, \neg \phi \implies \Delta )</td>
<td>( \Gamma, \phi \implies \Delta ) [ smash ] ( \Gamma \implies \neg \phi, \Delta )</td>
</tr>
<tr>
<td>and</td>
<td>( \Gamma, \phi, \psi \implies \Delta ) [ smash ] ( \Gamma, \phi \land \psi \implies \Delta )</td>
<td>( \Gamma \implies \phi, \Delta ) [ smash ] ( \Gamma \implies \psi, \Delta ) [ smash ] ( \Gamma \implies \phi \land \psi, \Delta )</td>
</tr>
<tr>
<td>or</td>
<td>( \Gamma, \phi \implies \Delta ) [ smash ] ( \Gamma, \psi \implies \Delta ) [ smash ] ( \Gamma, \phi</td>
<td>\psi \implies \Delta )</td>
</tr>
<tr>
<td>imp</td>
<td>( \Gamma \implies \phi, \Delta ) [ smash ] ( \Gamma, \psi \implies \Delta ) [ smash ] ( \Gamma, \phi \implies \psi \implies \Delta )</td>
<td>( \Gamma, \phi \implies \psi, \Delta ) [ smash ] ( \Gamma \implies \phi \implies \psi, \Delta )</td>
</tr>
</tbody>
</table>

\( \text{CLOSE} \) \[ smash ] \( \Gamma, \phi \implies \phi, \Delta \) \[ smash ] \( \text{TRUE} \) \[ smash ] \( \Gamma \implies \text{true}, \Delta \) \[ smash ] \( \text{FALSE} \) \[ smash ] \( \Gamma, \text{false} \implies \Delta \)
Justification of Rules

Compute rules by applying semantics definition of connectives
Justification of Rules

Compute rules by applying semantics definition of connectives

\[
\text{OR_RIGHT} \quad \Gamma \implies \phi, \psi, \Delta \\
\Gamma \implies \phi \mid \psi, \Delta
\]

Follows directly from semantics of sequents
Justification of Rules

Compute rules by applying semantics definition of connectives

**OR**

\[ \Gamma \Rightarrow \phi, \psi, \Delta \]

\[ \Gamma \Rightarrow \phi | \psi, \Delta \]

Follows directly from semantics of sequents

**AND**

\[ \Gamma \Rightarrow \phi, \Delta \]

\[ \Gamma \Rightarrow \psi, \Delta \]

\[ \Gamma \Rightarrow \phi & \psi, \Delta \]

\[ \Gamma \Rightarrow (\phi & \psi) | \Delta \quad \text{iff} \quad \Gamma \Rightarrow \phi | \Delta \quad \text{and} \quad \Gamma \Rightarrow \psi | \Delta \]

Distributivity of & over | and ->
Sequent Calculus Proofs

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

- find rule $R$ whose conclusion matches $G$
- instantiate $R$ such that conclusion identical to $G$
- recursively find proofs for resulting premisses $G_1, \ldots, G_r$
- 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
A Simple Proof

\[ \implies (A \land (A \rightarrow B)) \rightarrow B \]
A Simple Proof

\[ A \& (A \rightarrow B) \implies B \]

\[ \implies (A \& (A \rightarrow B)) \rightarrow B \]

By imp right

\[ \Gamma, \phi \implies \psi, \Delta \]

\[ \Gamma \implies \phi \rightarrow \psi, \Delta \]
A Simple Proof

\[ A, (A \rightarrow B) \implies B \]

\[ A \& (A \rightarrow B) \implies B \]

\[ \implies (A \& (A \rightarrow B)) \rightarrow B \]

By and left

\[ \Gamma, \phi, \psi \implies \Delta \]

\[ \Gamma, \phi \& \psi \implies \Delta \]
A Simple Proof

\[
\begin{align*}
A & \implies B, A \\
A, (A \implies B) & \implies B \\
A \land (A \implies B) & \implies B \\
\implies (A \land (A \implies B)) & \implies B
\end{align*}
\]

By \textbf{imp left} \quad
\[
\begin{array}{c}
\frac{
\Gamma \implies \phi, \Delta \quad \Gamma, \psi \implies \Delta
}{
\Gamma, \phi \implies \psi \implies \Delta
}\end{array}
\]
A Simple Proof

<table>
<thead>
<tr>
<th>∗</th>
<th>∗</th>
<th>∗</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \implies B, A )</td>
<td>( A, B \implies B )</td>
<td></td>
</tr>
<tr>
<td>( A, (A \implies B) \implies B )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A \land (A \implies B) \implies B )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \implies (A \land (A \implies B)) \implies B )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By close

\( \Gamma, \phi \implies \phi, \Delta \)
A Simple Proof

A proof is **closed**, if all its branches are closed.
Propositional Logic is insufficient

\( A \quad \text{ALL PERSONS ARE HAPPY} \)
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 possibility to talk about individuals
In particular, need to model objects, attributes, associations, etc.
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 possibility to talk about individuals
In particular, need to model objects, attributes, associations, etc.

⇒ First-Order Logic (FOL)