Software Engineering using Formal Methods
First-Order Logic

Wolfgang Ahrendt

26th September 2013
Install the KeY-Tool...

KeY used in Friday’s exercise

Requires: Java $\geq 5$

Follow instructions on course page, under:
⇒ Links, Papers, and Software

We recommend using Java Web Start:

- Start KeY with two clicks
  (you need to trust our self-signed certificate)
- Java Web Start installed with standard JDK/JRE
- Usually browsers know filetype.
  Otherwise open KeY.jnlp with javaws.

SEFM: First-Order Logic
Install the KeY-Tool...

KeY used in Friday’s exercise

Requires: Java $\geq 5$

Follow instructions on course page, under:
⇒ Links, Papers, and Software

We recommend using Java Web Start:

- Start KeY with two clicks
  (you need to trust our self-signed certificate)
- Java Web Start installed with standard JDK/JRE
- Usually browsers know filetype.
  Otherwise open KeY.jnlp with javaws.

If you want to install KeY locally instead, download from www.key-project.org. For this course, install version 1.6.x.
Motivation for Introducing First-Order Logic

1) we specify Java programs with Java Modeling Language (JML)

JML combines

- Java expressions
- First-Order Logic (FOL)
Motivation for Introducing First-Order Logic

1) we specify Java programs with Java Modeling Language (JML)

**JML combines**

- Java expressions
- First-Order Logic (FOL)

2) we verify Java programs using Dynamic Logic

**Dynamic Logic combines**

- First-Order Logic (FOL)
- Java programs
we introduce:

- FOL as a language
- calculus for proving FOL formulas
- KeY system as propositional, and first-order, prover (for now)
- (formal semantics: if time)
Signature

A first-order signature $\Sigma$ consists of

- a set $T_\Sigma$ of types
- a set $F_\Sigma$ of function symbols
- a set $P_\Sigma$ of predicate symbols
- a typing $\alpha_\Sigma$
First-Order Logic: Signature

Signature

A first-order signature $\Sigma$ consists of

- a set $T_\Sigma$ of types
- a set $F_\Sigma$ of function symbols
- a set $P_\Sigma$ of predicate symbols
- a typing $\alpha_\Sigma$

Intuitively, the typing $\alpha_\Sigma$ determines

- for each function and predicate symbol:
  - its arity, i.e., number of arguments
  - its argument types
- for each function symbol its result type.
A first-order signature $\Sigma$ consists of

- a set $T_\Sigma$ of types
- a set $F_\Sigma$ of function symbols
- a set $P_\Sigma$ of predicate symbols
- a typing $\alpha_\Sigma$

Intuitively, the typing $\alpha_\Sigma$ determines

- for each function and predicate symbol:
  - its arity, i.e., number of arguments
  - its argument types
- for each function symbol its result type.

Formally:

- $\alpha_\Sigma(p) \in T_\Sigma^*$ for all $p \in P_\Sigma$ (arity of $p$ is $|\alpha_\Sigma(p)|$)
- $\alpha_\Sigma(f) \in T_\Sigma^* \times T_\Sigma$ for all $f \in F_\Sigma$ (arity of $f$ is $|\alpha_\Sigma(f)| - 1$)
Example Signature 1 + Constants

\[ T_{\Sigma_1} = \{ \text{int} \}, \]
\[ F_{\Sigma_1} = \{ +, - \} \cup \{ ..., -2, -1, 0, 1, 2, ... \}, \]
\[ P_{\Sigma_1} = \{ < \} \]
Example Signature 1 + Constants

\[ T_{\Sigma_1} = \{ \text{int} \}, \]
\[ F_{\Sigma_1} = \{ +, - \} \cup \{ \ldots, -2, -1, 0, 1, 2, \ldots \}, \]
\[ P_{\Sigma_1} = \{ < \} \]

\[ \alpha_{\Sigma_1}(<) = (\text{int}, \text{int}) \]
\[ \alpha_{\Sigma_1}(+) = \alpha_{\Sigma_1}(-) = (\text{int}, \text{int}, \text{int}) \]
\[ \alpha_{\Sigma_1}(0) = \alpha_{\Sigma_1}(1) = \alpha_{\Sigma_1}(-1) = \ldots = (\text{int}) \]
Example Signature 1 + Constants

\[ T_{\Sigma_1} = \{\text{int}\}, \]
\[ F_{\Sigma_1} = \{+, -\} \cup \{..., -2, -1, 0, 1, 2, ...\}, \]
\[ P_{\Sigma_1} = \{<\} \]

\[ \alpha_{\Sigma_1}(<) = (\text{int}, \text{int}) \]
\[ \alpha_{\Sigma_1}(+) = \alpha_{\Sigma_1}(-) = (\text{int}, \text{int}, \text{int}) \]
\[ \alpha_{\Sigma_1}(0) = \alpha_{\Sigma_1}(1) = \alpha_{\Sigma_1}(-1) = ... = (\text{int}) \]

**Constant Symbols**

A function symbol \( f \) with \( |\alpha_{\Sigma_1}(f)| = 1 \) (i.e., with arity 0) is called *constant symbol*. 
Example Signature 1 + Constants

\[ T_{\Sigma_1} = \{\text{int}\}, \]
\[ F_{\Sigma_1} = \{+, -\} \cup \{\ldots, -2, -1, 0, 1, 2, \ldots\}, \]
\[ P_{\Sigma_1} = \{<\} \]

\[ \alpha_{\Sigma_1}(<) = (\text{int}, \text{int}) \]
\[ \alpha_{\Sigma_1}(+) = \alpha_{\Sigma_1}(-) = (\text{int}, \text{int}, \text{int}) \]
\[ \alpha_{\Sigma_1}(0) = \alpha_{\Sigma_1}(1) = \alpha_{\Sigma_1}(-1) = \ldots = (\text{int}) \]

Constant Symbols

A function symbol \( f \) with \( |\alpha_{\Sigma_1}(f)| = 1 \) (i.e., with arity 0) is called constant symbol.

Here, the constant symbols are: \( \ldots, -2, -1, 0, 1, 2, \ldots \)
Type declaration of signature symbols

- Write $\tau \ x$; to declare variable $x$ of type $\tau$
- Write $p(\tau_1, \ldots, \tau_r)$; for $\alpha(p) = (\tau_1, \ldots, \tau_r)$
- Write $\tau \ f(\tau_1, \ldots, \tau_r)$; for $\alpha(f) = (\tau_1, \ldots, \tau_r, \tau)$

$r = 0$ is allowed, then write $f$ instead of $f()$, etc.
Type declaration of signature symbols

- Write $\tau \ x$; to declare variable $x$ of type $\tau$
- Write $p(\tau_1, \ldots, \tau_r)$; for $\alpha(p) = (\tau_1, \ldots, \tau_r)$
- Write $\tau \ f(\tau_1, \ldots, \tau_r)$; for $\alpha(f) = (\tau_1, \ldots, \tau_r, \tau)$

$r = 0$ is allowed, then write $f$ instead of $f()$, etc.

Example

- Variables: integerArray a; int i;
- Predicate Symbols: isEmpty(List); alertOn;
- Function Symbols: int arrayLookupup(int); Object o;
typing of Signature 1:

\[ \alpha_{\Sigma_1}(\langle \rangle) = (\text{int}, \text{int}) \]
\[ \alpha_{\Sigma_1}(+) = \alpha_{\Sigma_1}(-) = (\text{int}, \text{int}, \text{int}) \]
\[ \alpha_{\Sigma_1}(0) = \alpha_{\Sigma_1}(1) = \alpha_{\Sigma_1}(-1) = \ldots = (\text{int}) \]

can alternatively be written as:

\(<(\text{int}, \text{int});\>
\int + (\text{int}, \text{int});\>
\int 0; \int 1; \int -1; \ldots
Example Signature 2

\[ T_{\Sigma_2} = \{ \text{int}, \text{LinkedIntList} \}, \]
\[ F_{\Sigma_2} = \{ \text{null}, \text{new}, \text{elem}, \text{next} \} \cup \{ \ldots, -2, -1, 0, 1, 2, \ldots \} \]
\[ P_{\Sigma_2} = \{ \} \]
Example Signature 2

\[ T_{\Sigma_2} = \{ \text{int, LinkedIntList} \}, \]
\[ F_{\Sigma_2} = \{ \text{null, new, elem, next} \} \cup \{..., -2, -1, 0, 1, 2,...\} \]
\[ P_{\Sigma_2} = \{\} \]

intuitively, elem and next model fields of LinkedIntList objects
Example Signature 2

\[ T_{\Sigma_2} = \{ \text{int, LinkedIntList} \}, \]
\[ F_{\Sigma_2} = \{ \text{null, new, elem, next} \} \cup \{ ..., -2, -1, 0, 1, 2, ... \} \]
\[ P_{\Sigma_2} = \{ \} \]

intuitively, elem and next model fields of LinkedIntList objects

type declarations:

```java
LinkedIntList null;
LinkedIntList new(int, LinkedIntList);
int elem(LinkedIntList);
LinkedIntList next(LinkedIntList);
and as before:
int 0; int 1; int -1; ...
```
$T_{\Sigma_2} = \{\text{int, LinkedIntList}\}$,
$F_{\Sigma_2} = \{\text{null, new, elem, next}\} \cup \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$P_{\Sigma_2} = \{\}$

intuitively, elem and next model fields of LinkedIntList objects

type declarations:

LinkedIntList null;
LinkedIntList new(int, LinkedIntList);
int elem(LinkedIntList);
LinkedIntList next(LinkedIntList);
Example Signature 2

\[ T_{\Sigma_2} = \{\text{int, LinkedIntList}\}, \]
\[ F_{\Sigma_2} = \{\text{null, new, elem, next}\} \cup \{\ldots,-2,-1,0,1,2,\ldots\} \]
\[ P_{\Sigma_2} = \{\} \]

intuitively, elem and next model fields of LinkedIntList objects

type declarations:

LinkedIntList null;
LinkedIntList new(int, LinkedIntList);
int elem(LinkedIntList);
LinkedIntList next(LinkedIntList);

and as before:
int 0; int 1; int -1; ...
First-Order Terms

We assume a set \( V \) of variables \((V \cap (F_\Sigma \cup P_\Sigma) = \emptyset)\).
Each \( v \in V \) has a unique type \( \alpha_\Sigma(v) \in T_\Sigma \).
First-Order Terms

We assume a set $V$ of variables ($V \cap (F_{\Sigma} \cup P_{\Sigma}) = \emptyset$). Each $v \in V$ has a unique type $\alpha_{\Sigma}(v) \in T_{\Sigma}$.

Terms are defined recursively:

**Terms**

A first-order term of type $\tau \in T_{\Sigma}$
- is either a variable of type $\tau$, or
- has the form $f(t_1, \ldots, t_n)$, where $f \in F_{\Sigma}$ has result type $\tau$, and each $t_i$ is term of the correct type, following the typing $\alpha_{\Sigma}$ of $f$. 
First-Order Terms

We assume a set \( V \) of variables \( (V \cap (F_\Sigma \cup P_\Sigma) = \emptyset) \).
Each \( v \in V \) has a unique type \( \alpha_\Sigma(v) \in T_\Sigma \).

Terms are defined recursively:

**Terms**

A first-order term of type \( \tau \in T_\Sigma \)
- is either a variable of type \( \tau \), or
- has the form \( f(t_1, \ldots, t_n) \),
  where \( f \in F_\Sigma \) has result type \( \tau \), and each \( t_i \) is term of the correct type, following the typing \( \alpha_\Sigma \) of \( f \).

If \( f \) is a constant symbol, the term is written \( f \), instead of \( f() \).
Terms over Signature 1

example terms over $\Sigma_1$:
(assume variables int $v_1$; int $v_2$;)

- $-2 + 99$
- $7 - 8$
- $(7 - 8) + 1$
- $(-v_1 - 8) + v_2$
Terms over Signature 1

example terms over $\Sigma_1$:
(assume variables int $v_1$; int $v_2$)

- $-7$
- $+(-2, 99)$
- $-(7, 8)$
- $+(-7, 8, 1)$
- $+(-v_1, 8, v_2)$
example terms over $\Sigma_1$:
(assume variables int $v_1$; int $v_2$;)

-7
+(-2, 99)
-(7, 8)
+(-(7, 8), 1)
+(-(v_1, 8), v_2)

some variants of FOL allow infix notation of functions:

-2 + 99
7 - 8
(7 - 8) + 1
(v_1 - 8) + v_2
Terms over Signature 2

example terms over $\Sigma_2$:
(assume variables LinkedIntList o; int v;)

for first-order functions modeling object fields,
we allow dotted postfix notation:

```
new (13, null).
elem o.
next (next o).
```
example terms over $\Sigma_2$:
(assume variables LinkedListIntList o; int v;)

-7
null
new(13, null)
elem(new(13, null))
next(next(o))
example terms over $\Sigma_2$:
(assume variables LinkedIntList o; int v;)

-7
null
new(13, null)
elem(new(13, null))
next(next(o))

for first-order functions modeling object fields, we allow dotted postfix notation:

new(13, null).elem
o.next.next
Given a signature $\Sigma$. An atomic formula has either of the forms

- $true$
- $false$
- $t_1 = t_2$ ("equality"), where $t_1$ and $t_2$ are first-order terms of the same type.
- $p(t_1, \ldots, t_n)$ ("predicate"), where $p \in P_\Sigma$, and each $t_i$ is term of the correct type, following the typing $\alpha_\Sigma$ of $p$. 
example formulas over $\Sigma_1$:
(assume variable int $\nu$;)

▶ $7 = 8$
▶ $7 < 8$
▶ $-2 < 9$
▶ $\nu < (\nu + 1)$
example formulas over $\Sigma_1$:
(assume variable int $v$;)

- $7 = 8$
- $7 < 8$
- $-2 - v < 99$
- $v < (v + 1)$
example formulas over $\Sigma_2$:
(assume variables LinkedIntList o; int v;)

\[\new\(13\),null = null\]
\[\elem\(\new\(13\),null\) = 13\]
\[\next\(\new\(13\),null\) = null\]
\[\next\(\next(o)\) = o\]
Atomic Formulas over Signature 2

example formulas over $\Sigma_2$:
(assume variables LinkedIntList o; int v;)

- $\text{new}(13, \text{null}) = \text{null}$
- $\text{elem}(\text{new}(13, \text{null})) = 13$
- $\text{next}(\text{new}(13, \text{null})) = \text{null}$
- $\text{next}(\text{next}(o)) = o$
First-order Formulas

Formulas

- each atomic formula is a formula
- with $\phi$ and $\psi$ formulas, $x$ a variable, and $\tau$ a type, the following are also formulas:
  - $\neg \phi$ ("not $\phi$")
  - $\phi \land \psi$ ("$\phi$ and $\psi$")
  - $\phi \lor \psi$ ("$\phi$ or $\psi$")
  - $\phi \rightarrow \psi$ ("$\phi$ implies $\psi$")
  - $\phi \leftrightarrow \psi$ ("$\phi$ is equivalent to $\psi$")
First-order Formulas

Formulas

- each atomic formula is a formula
- with $\phi$ and $\psi$ formulas, $x$ a variable, and $\tau$ a type, the following are also formulas:
  - $\neg \phi$ ("not $\phi$")
  - $\phi \land \psi$ ("$\phi$ and $\psi$")
  - $\phi \lor \psi$ ("$\phi$ or $\psi$")
  - $\phi \rightarrow \psi$ ("$\phi$ implies $\psi$")
  - $\phi \leftrightarrow \psi$ ("$\phi$ is equivalent to $\psi$")
  - $\forall \tau \ x; \ \phi$ ("for all $x$ of type $\tau$ holds $\phi$")
First-order Formulas

Formulas

- each atomic formula is a formula
- with \( \phi \) and \( \psi \) formulas, \( x \) a variable, and \( \tau \) a type, the following are also formulas:
  - \( \neg \phi \) ("not \( \phi \")
  - \( \phi \land \psi \) ("\( \phi \) and \( \psi \")
  - \( \phi \lor \psi \) ("\( \phi \) or \( \psi \")
  - \( \phi \rightarrow \psi \) ("\( \phi \) implies \( \psi \")
  - \( \phi \leftrightarrow \psi \) ("\( \phi \) is equivalent to \( \psi \")
  - \( \forall \tau \; x; \; \phi \) ("for all \( x \) of type \( \tau \) holds \( \phi \")
  - \( \exists \tau \; x; \; \phi \) ("there exists an \( x \) of type \( \tau \) such that \( \phi \")

In \( \forall \tau \; x; \; \phi \) and \( \exists \tau \; x; \; \phi \) the variable \( x \) is 'bound' (i.e., 'not free'). Formulas with no free variable are 'closed'.

SEFM: First-Order Logic

CHALMERS
First-order Formulas

Formulas

- each atomic formula is a formula
- with $\phi$ and $\psi$ formulas, $x$ a variable, and $\tau$ a type, the following are also formulas:
  - $\neg \phi$ ("not $\phi$")
  - $\phi \land \psi$ ("$\phi$ and $\psi$")
  - $\phi \lor \psi$ ("$\phi$ or $\psi$")
  - $\phi \rightarrow \psi$ ("$\phi$ implies $\psi$")
  - $\phi \leftrightarrow \psi$ ("$\phi$ is equivalent to $\psi$")
  - $\forall \tau \ x; \ \phi$ ("for all $x$ of type $\tau$ holds $\phi$")
  - $\exists \tau \ x; \ \phi$ ("there exists an $x$ of type $\tau$ such that $\phi$")

In $\forall \tau \ x; \ \phi$ and $\exists \tau \ x; \ \phi$ the variable $x$ is ‘bound’ (i.e., ‘not free’).
First-order Formulas

Formulas

- each atomic formula is a formula
- with $\phi$ and $\psi$ formulas, $x$ a variable, and $\tau$ a type, the following are also formulas:
  - $\neg \phi$ ("not $\phi$")
  - $\phi \land \psi$ ("$\phi$ and $\psi$")
  - $\phi \lor \psi$ ("$\phi$ or $\psi$")
  - $\phi \rightarrow \psi$ ("$\phi$ implies $\psi$")
  - $\phi \leftrightarrow \psi$ ("$\phi$ is equivalent to $\psi$")
  - $\forall \tau \; x; \; \phi$ ("for all $x$ of type $\tau$ holds $\phi$")
  - $\exists \tau \; x; \; \phi$ ("there exists an $x$ of type $\tau$ such that $\phi$")

In $\forall \tau \; x; \; \phi$ and $\exists \tau \; x; \; \phi$ the variable $x$ is ‘bound’ (i.e., ‘not free’).
Formulas with no free variable are ‘closed’.
(signatures/types left out here)

Example (There are at least two elements)
(signatures/types left out here)

Example (There are at least two elements)
\[ \exists x, y; \neg (x = y) \]
First-order Formulas: Examples

(signatures/types left out here)

Example (Strict partial order)
First-order Formulas: Examples

Example (Strict partial order)

Irreflexivity \( \forall x; \neg(x < x) \)
Asymmetry \( \forall x; \forall y; (x < y \rightarrow \neg(y < x)) \)
Transitivity \( \forall x; \forall y; \forall z; \\
\quad (x < y \land y < z \rightarrow x < z) \)
Example (Strict partial order)

Irreflexivity \( \forall x; \neg(x < x) \)
Asymmetry \( \forall x; \forall y; (x < y \rightarrow \neg(y < x)) \)
Transitivity \( \forall x; \forall y; \forall z; \\
(x < y \land y < z \rightarrow x < z) \)

(is any of the three formulas redundant?)
Domain

A domain $D$ is a set of elements which are (potentially) the *meaning* of terms and variables.
Semantics (briefly here, more thorough later)

**Domain**

A domain $\mathcal{D}$ is a set of elements which are (potentially) the meaning of terms and variables.

**Interpretation**

An interpretation $\mathcal{I}$ (over $\mathcal{D}$) assigns meaning to the symbols in $F_{\Sigma} \cup P_{\Sigma}$ (assigning functions to function symbols, relations to predicate symbols).
## Semantics (briefly here, more thorough later)

### Domain

A domain $\mathcal{D}$ is a set of elements which are (potentially) the *meaning* of terms and variables.

### Interpretation

An interpretation $\mathcal{I}$ (over $\mathcal{D}$) assigns *meaning* to the symbols in $F_\Sigma \cup P_\Sigma$ (assigning functions to function symbols, relations to predicate symbols).

### Valuation

In a given $\mathcal{D}$ and $\mathcal{I}$, a closed formula evaluates to either $T$ or $F$. 
## Semantics (briefly here, more thorough later)

### Domain
A domain $\mathcal{D}$ is a set of elements which are (potentially) the *meaning* of terms and variables.

### Interpretation
An interpretation $\mathcal{I}$ (over $\mathcal{D}$) assigns *meaning* to the symbols in $F_\Sigma \cup P_\Sigma$ (assigning functions to function symbols, relations to predicate symbols).

### Valuation
In a given $\mathcal{D}$ and $\mathcal{I}$, a closed formula evaluates to either $T$ or $F$.

### Validity
A closed formula is *valid* if it evaluates to $T$ in all $\mathcal{D}$ and $\mathcal{I}$.
Semantics (briefly here, more thorough later)

**Domain**
A domain $\mathcal{D}$ is a set of elements which are (potentially) the meaning of terms and variables.

**Interpretation**
An interpretation $\mathcal{I}$ (over $\mathcal{D}$) assigns meaning to the symbols in $F_\Sigma \cup P_\Sigma$ (assigning functions to function symbols, relations to predicate symbols).

**Valuation**
In a given $\mathcal{D}$ and $\mathcal{I}$, a closed formula evaluates to either $T$ or $F$.

**Validity**
A closed formula is valid if it evaluates to $T$ in all $\mathcal{D}$ and $\mathcal{I}$.

In the context of specification/verification of programs: each $(\mathcal{D}, \mathcal{I})$ is called a ‘state’.
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:
Useful Valid Formulas

Let \( \phi \) and \( \psi \) be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

\[
\neg (\phi \land \psi) \iff \neg \phi \lor \neg \psi
\]

\[
\neg (\phi \lor \psi) \iff \neg \phi \land \neg \psi
\]

\[
(\text{true} \land \phi) \iff \phi
\]

\[
(\text{false} \lor \phi) \iff \phi
\]

\[
\neg (\text{false} \land \phi)
\]

\[
(\phi \to \psi) \iff (\neg \phi \lor \psi)
\]

\[
(\phi \to \text{true}) \iff \phi
\]

\[
\text{false} \to \phi \iff \phi
\]

\[
(\text{true} \to \phi) \iff \phi
\]

\[
(\phi \to \text{false}) \iff \neg \phi
\]
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid of not).

The following formulas are valid:

$\neg (\phi \land \psi) \leftrightarrow \neg \phi \lor \neg \psi$
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi$
- $\neg(\phi \lor \psi) \iff$
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg (\phi \land \psi) \iff \neg \phi \lor \neg \psi$
- $\neg (\phi \lor \psi) \iff \neg \phi \land \neg \psi$

SEFM: First-Order Logic
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi$
- $\neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi$
- $(true \land \phi) \iff$

$false \lor \phi \iff \neg(\neg\phi \land \phi)$
$true \rightarrow \phi \iff \phi$
$false \rightarrow \phi \iff \neg\phi$
$true \rightarrow (\phi \rightarrow false) \iff \neg\phi$
Useful Valid Formulas

Let \( \phi \) and \( \psi \) be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

1. \( \neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi \)
2. \( \neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi \)
3. \( (\text{true} \land \phi) \iff \phi \)
4. \( (\phi \rightarrow \psi) \iff (\neg\phi \lor \psi) \)
5. \( \phi \rightarrow \text{true} \)
6. \( \text{false} \rightarrow \phi \)
7. \( (\text{true} \rightarrow \phi) \iff \phi \)
8. \( (\phi \rightarrow \text{false}) \iff \neg\phi \)
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg(\phi \land \psi) \leftrightarrow \neg\phi \lor \neg\psi$
- $\neg(\phi \lor \psi) \leftrightarrow \neg\phi \land \neg\psi$
- $true \land \phi \leftrightarrow \phi$
- $false \lor \phi \leftrightarrow$
- $\phi \rightarrow$
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi$
- $\neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi$
- $(true \land \phi) \iff \phi$
- $(false \lor \phi) \iff \phi$
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

\begin{itemize}
  \item $\neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi$
  \item $\neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi$
  \item $(true \land \phi) \iff \phi$
  \item $(false \lor \phi) \iff \phi$
  \item $true \lor \phi$
  \item $false \land \phi$
\end{itemize}
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi$
- $\neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi$
- $(true \land \phi) \iff \phi$
- $(false \lor \phi) \iff \phi$
- $true \lor \phi$
- $false \land \phi$
- $\neg(false \land \phi)$
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi$
- $\neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi$
- $(true \land \phi) \iff \phi$
- $(false \lor \phi) \iff \phi$
- $true \lor \phi$
- $false \land \phi$
- $(false \land \phi)$
- $(\phi \rightarrow \psi) \iff$
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi$
- $\neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi$
- $(true \land \phi) \iff \phi$
- $(false \lor \phi) \iff \phi$
- $true \lor \phi$
- $false \land \phi$
- $\neg(false \land \phi)$
- $(\phi \to \psi) \iff (\neg\phi \lor \psi)$
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid of not).

The following formulas are valid:

1. \[ \neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi \]
2. \[ \neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi \]
3. \[ (\text{true} \land \phi) \iff \phi \]
4. \[ (\text{false} \lor \phi) \iff \phi \]
5. \[ \text{true} \lor \phi \]
6. \[ \neg(\text{false} \land \phi) \]
7. \[ (\phi \to \psi) \iff (\neg\phi \lor \psi) \]
8. \[ \phi \to \text{true} \]
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid of not).

The following formulas are valid:

- $\neg (\phi \land \psi) \iff \neg \phi \lor \neg \psi$
- $\neg (\phi \lor \psi) \iff \neg \phi \land \neg \psi$
- $(\text{true} \land \phi) \iff \phi$
- $(\text{false} \lor \phi) \iff \phi$
- $\text{true} \lor \phi$
- $\neg (\text{false} \land \phi)$
- $(\phi \rightarrow \psi) \iff (\neg \phi \lor \psi)$
- $\phi \rightarrow \text{true}$
- $\text{false} \rightarrow \phi$
Useful Valid Formulas

Let \( \phi \) and \( \psi \) be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- \( \neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi \)
- \( \neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi \)
- \( (true \land \phi) \iff \phi \)
- \( (false \lor \phi) \iff \phi \)
- \( true \lor \phi \)
- \( \neg(false \land \phi) \)
- \( (\phi \to \psi) \iff (\neg\phi \lor \psi) \)
- \( \phi \to true \)
- \( false \to \phi \)
- \( (true \to \phi) \iff \)

SEFM: First-Order Logic
Useful Valid Formulas

Let \( \phi \) and \( \psi \) be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

1. \( \neg (\phi \land \psi) \iff \neg \phi \lor \neg \psi \)
2. \( \neg (\phi \lor \psi) \iff \neg \phi \land \neg \psi \)
3. \( (\text{true} \land \phi) \iff \phi \)
4. \( (\text{false} \lor \phi) \iff \phi \)
5. \( \text{true} \lor \phi \)
6. \( \neg (\text{false} \land \phi) \)
7. \( (\phi \rightarrow \psi) \iff (\neg \phi \lor \psi) \)
8. \( \phi \rightarrow \text{true} \)
9. \( \text{false} \rightarrow \phi \)
10. \( (\text{true} \rightarrow \phi) \iff \phi \)
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg (\phi \land \psi) \iff \neg \phi \lor \neg \psi$
- $\neg (\phi \lor \psi) \iff \neg \phi \land \neg \psi$
- $(true \land \phi) \iff \phi$
- $(false \lor \phi) \iff \phi$
- $true \lor \phi$
- $false \land \phi$
- $\neg (false \land \phi)$
- $(\phi \to \psi) \iff (\neg \phi \lor \psi)$
- $\phi \to true$
- $false \to \phi$
- $(true \to \phi) \iff \phi$
- $(\phi \to false) \iff \phi$
Useful Valid Formulas

Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid or not).

The following formulas are valid:

- $\neg (\phi \land \psi) \iff \neg \phi \lor \neg \psi$
- $\neg (\phi \lor \psi) \iff \neg \phi \land \neg \psi$
- $(true \land \phi) \iff \phi$
- $(false \lor \phi) \iff \phi$
- $true \lor \phi$
- $\neg (false \land \phi)$
- $(\phi \rightarrow \psi) \iff (\neg \phi \lor \psi)$
- $\phi \rightarrow true$
- $false \rightarrow \phi$
- $(true \rightarrow \phi) \iff \phi$
- $(\phi \rightarrow false) \iff \neg \phi$
Useful Valid Formulas

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

\[
\neg (\exists \tau x; \phi) \leftrightarrow \forall \tau x; \neg \phi
\]

\[
\neg (\forall \tau x; \phi) \leftrightarrow \exists \tau x; \neg \phi
\]

\[
(\forall \tau x; \phi \land \psi) \leftrightarrow (\forall \tau x; \phi) \land (\forall \tau x; \psi)
\]

\[
(\exists \tau x; \phi \lor \psi) \leftrightarrow (\exists \tau x; \phi) \lor (\exists \tau x; \psi)
\]

Are the following formulas also valid?

\[
(\forall \tau x; \phi \lor \psi) \leftrightarrow (\forall \tau x; \phi) \lor (\forall \tau x; \psi)
\]

\[
(\exists \tau x; \phi \land \psi) \leftrightarrow (\exists \tau x; \phi) \land (\exists \tau x; \psi)
\]
Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

- $\neg(\exists \tau \ x; \ \phi) \iff \forall \tau \ x; \ \neg \phi$
- $\neg(\forall \tau \ x; \ \phi) \iff \exists \tau \ x; \ \neg \phi$
- $(\forall \tau \ x; \ \phi \land \psi) \iff (\forall \tau \ x; \ \phi) \land (\forall \tau \ x; \ \psi)$
- $(\exists \tau \ x; \ \phi \lor \psi) \iff (\exists \tau \ x; \ \phi) \lor (\exists \tau \ x; \ \psi)$
Useful Valid Formulas

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

$\neg (\exists \tau \ x; \phi) \leftrightarrow \forall \tau \ x; \neg \phi$

Are the following formulas also valid?

$\neg (\forall \tau \ x; \phi) \leftrightarrow \exists \tau \ x; \neg \phi$

$\exists \tau \ x; \phi \land \psi \leftrightarrow \exists \tau \ x; \phi \land \exists \tau \ x; \psi$

SEFM: First-Order Logic
Useful Valid Formulas

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

- $\neg(\exists \tau \ x; \ \phi) \leftrightarrow \forall \tau \ x; \ \neg \phi$
- $\neg(\forall \tau \ x; \ \phi) \leftrightarrow$
Useful Valid Formulas

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

- $\neg(\exists \tau \; x; \; \phi) \leftrightarrow \forall \tau \; x; \; \neg \phi$
- $\neg(\forall \tau \; x; \; \phi) \leftrightarrow \exists \tau \; x; \; \neg \phi$
Useful Valid Formulas

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

- $\neg(\exists \tau \; x; \; \phi) \leftrightarrow \forall \tau \; x; \; \neg \phi$
- $\neg(\forall \tau \; x; \; \phi) \leftrightarrow \exists \tau \; x; \; \neg \phi$
- $(\forall \tau \; x; \; \phi \land \psi) \leftrightarrow$
Useful Valid Formulas

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

- $\neg(\exists \tau \ x; \ \phi) \leftrightarrow \forall \tau \ x; \ \neg \phi$
- $\neg(\forall \tau \ x; \ \phi) \leftrightarrow \exists \tau \ x; \ \neg \phi$
- $(\forall \tau \ x; \ \phi \land \psi) \leftrightarrow (\forall \tau \ x; \ \phi) \land (\forall \tau \ x; \ \psi)$
Useful Valid Formulas

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

1. $\neg (\exists \tau \ x; \ \phi) \leftrightarrow \forall \tau \ x; \ \neg \phi$
2. $\neg (\forall \tau \ x; \ \phi) \leftrightarrow \exists \tau \ x; \ \neg \phi$
3. $(\forall \tau \ x; \ \phi \land \psi) \leftrightarrow (\forall \tau \ x; \ \phi) \land (\forall \tau \ x; \ \psi)$
4. $(\exists \tau \ x; \ \phi \lor \psi) \leftrightarrow$
Useful Valid Formulas

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

$\neg(\exists \tau x; \phi) \iff \forall \tau x; \neg\phi$

$\neg(\forall \tau x; \phi) \iff \exists \tau x; \neg\phi$

$(\forall \tau x; \phi \land \psi) \iff (\forall \tau x; \phi) \land (\forall \tau x; \psi)$

$(\exists \tau x; \phi \lor \psi) \iff (\exists \tau x; \phi) \lor (\exists \tau x; \psi)$
Useful Valid Formulas

Assume that \( x \) is the only variable which may appear freely in \( \phi \) or \( \psi \).

The following formulas are valid:

- \( \neg(\exists \, \tau \; x; \, \phi) \leftrightarrow \forall \, \tau \; x; \, \neg \phi \)
- \( \neg(\forall \, \tau \; x; \, \phi) \leftrightarrow \exists \, \tau \; x; \, \neg \phi \)
- \( (\forall \, \tau \; x; \, \phi \land \psi) \leftrightarrow (\forall \, \tau \; x; \, \phi) \land (\forall \, \tau \; x; \, \psi) \)
- \( (\exists \, \tau \; x; \, \phi \lor \psi) \leftrightarrow (\exists \, \tau \; x; \, \phi) \lor (\exists \, \tau \; x; \, \psi) \)

Are the following formulas also valid?
Useful Valid Formulas

Assume that \( x \) is the only variable which may appear freely in \( \phi \) or \( \psi \).

The following formulas are valid:

- \( \neg (\exists \tau \ x; \ \phi) \leftrightarrow \forall \tau \ x; \ \neg \phi \)
- \( \neg (\forall \tau \ x; \ \phi) \leftrightarrow \exists \tau \ x; \ \neg \phi \)
- \( (\forall \tau \ x; \ \phi \land \psi) \leftrightarrow (\forall \tau \ x; \ \phi) \land (\forall \tau \ x; \ \psi) \)
- \( (\exists \tau \ x; \ \phi \lor \psi) \leftrightarrow (\exists \tau \ x; \ \phi) \lor (\exists \tau \ x; \ \psi) \)

Are the following formulas also valid?

- \( (\forall \tau \ x; \ \phi \lor \psi) \leftrightarrow (\forall \tau \ x; \ \phi) \lor (\forall \tau \ x; \ \psi) \)
Useful Valid Formulas

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$.

The following formulas are valid:

- $\neg(\exists \tau x; \phi) \leftrightarrow \forall \tau x; \neg \phi$
- $\neg(\forall \tau x; \phi) \leftrightarrow \exists \tau x; \neg \phi$
- $(\forall \tau x; \phi \land \psi) \leftrightarrow (\forall \tau x; \phi) \land (\forall \tau x; \psi)$
- $(\exists \tau x; \phi \lor \psi) \leftrightarrow (\exists \tau x; \phi) \lor (\exists \tau x; \psi)$

Are the following formulas also valid?

- $(\forall \tau x; \phi \lor \psi) \leftrightarrow (\forall \tau x; \phi) \lor (\forall \tau x; \psi)$
- $(\exists \tau x; \phi \land \psi) \leftrightarrow (\exists \tau x; \phi) \land (\exists \tau x; \psi)$
## Remark on Concrete Syntax

<table>
<thead>
<tr>
<th></th>
<th>Text book</th>
<th>SPIN</th>
<th>KeY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation</td>
<td>(\neg)</td>
<td>(!)</td>
<td>(!)</td>
</tr>
<tr>
<td>Conjunction</td>
<td>(\wedge)</td>
<td>(&amp;&amp;)</td>
<td>&amp;</td>
</tr>
<tr>
<td>Disjunction</td>
<td>(\vee)</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>Implication</td>
<td>(\rightarrow, \supset)</td>
<td>(\rightarrow)</td>
<td>(\rightarrow)</td>
</tr>
<tr>
<td>Equivalence</td>
<td>(\Leftrightarrow)</td>
<td>(\leftrightarrow)</td>
<td>(\leftrightarrow)</td>
</tr>
<tr>
<td>Universal Quantifier</td>
<td>(\forall x; \phi)</td>
<td>n/a</td>
<td>\text{	extbackslash forall} (\tau x; \phi)</td>
</tr>
<tr>
<td>Existential Quantifier</td>
<td>(\exists x; \phi)</td>
<td>n/a</td>
<td>\text{	extbackslash exists} (\tau x; \phi)</td>
</tr>
<tr>
<td>Value equality</td>
<td>(=)</td>
<td>(==)</td>
<td>(=)</td>
</tr>
</tbody>
</table>
Part I

Sequent Calculus for FOL
Reasoning by Syntactic Transformation

Prove Validity of $\phi$ by syntactic transformation of $\phi$
Reasoning by Syntactic Transformation

Prove Validity of $\phi$ by syntactic transformation of $\phi$

Logic Calculus: **Sequent Calculus** based on notion of *sequent*:

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

Antecedent \quad Succedent
Reasoning by Syntactic Transformation

Prove Validity of $\phi$ by **syntactic** transformation of $\phi$

Logic Calculus: **Sequent Calculus** based on notion of **sequent**:

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

Antecedent $\Rightarrow$ Succedent

has same meaning as

$$(\psi_1 \land \cdots \land \psi_m) \rightarrow (\phi_1 \lor \cdots \lor \phi_n)$$
Reasoning by Syntactic Transformation

Prove Validity of $\phi$ by syntactic transformation of $\phi$.

Logic Calculus: Sequent Calculus based on notion of sequent:

\[
\begin{array}{c}
\psi_1, \ldots, \psi_m \quad \Rightarrow \\
\text{Antecedent}
\end{array}
\begin{array}{c}
\phi_1, \ldots, \phi_n \\
\text{Succedent}
\end{array}
\]

has same meaning as

\[
(\psi_1 \land \cdots \land \psi_m) \quad \Rightarrow \\
(\phi_1 \lor \cdots \lor \phi_n)
\]

which has (for closed formulas $\psi_i, \phi_i$) same meaning as

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

\[ \psi_1, \ldots, \psi_m \implies \phi_1, \ldots, \phi_n \]

Consider antecedent/succedent as sets of formulas, may be empty
Notation for Sequents

\[ \psi_1, \ldots, \psi_m \implies \phi_1, \ldots, \phi_n \]

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

**Schema Variables**

\( \phi, \psi, \ldots \) match formulas, \( \Gamma, \Delta, \ldots \) match sets of formulas

Characterize infinitely many sequents with single schematic sequent, e.g.,

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

Matches any sequent with occurrence of conjunction in succedent

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

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

Write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible.

RuleName

\[ \Gamma_1 \Rightarrow \Delta_1 \quad \ldots \quad \Gamma_r \Rightarrow \Delta_r \]

Premisses

\[ \Gamma \Rightarrow \Delta \]

Conclusion

Meaning: For proving the Conclusion, it suffices to prove all Premisses.

Example andRight

\[ \Gamma = \Rightarrow \varphi, \quad \Delta \Gamma = \Rightarrow \psi, \quad \Delta = \Rightarrow \varphi \land \psi, \quad \Delta \]

Admissible to have no premisses (iff conclusion is valid, e.g., axiom)

A rule is sound (correct) iff the validity of its premisses implies the validity of its conclusion.
Sequent Calculus Rules

Write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible

RuleName

\[ \Gamma_1 \Rightarrow \Delta_1 \quad \cdots \quad \Gamma_r \Rightarrow \Delta_r \]

Premisses

\[ \Gamma \Rightarrow \Delta \]

Conclusion

Meaning: For proving the Conclusion, it suffices to prove all Premisses.
Write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible

**RuleName**

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

**Premisses**

**Conclusion**

Meaning: For proving the Conclusion, it suffices to prove all Premisses.

**Example**

andRight

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

\[
\Gamma \Rightarrow \phi \land \psi, \Delta
\]
Sequent Calculus Rules

Write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible

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

RuleName

Premisses

Meaning: For proving the Conclusion, it suffices to prove all Premisses.

Example

\[
\text{andRight} \quad \frac{\Gamma \Rightarrow \phi, \Delta \quad \Gamma \Rightarrow \psi, \Delta}{\Gamma \Rightarrow \phi \land \psi, \Delta}
\]

Admissible to have no premisses (iff conclusion is valid, e.g., axiom)
Sequent Calculus Rules

Write syntactic transformation schema for sequents that reflects semantics of connectives as closely as possible

\[
\begin{align*}
\text{RuleName} & \quad \text{Premisses} \\
& \quad \Gamma_1 \Rightarrow \Delta_1 \quad \cdots \quad \Gamma_r \Rightarrow \Delta_r \\
& \quad \Gamma \Rightarrow \Delta \\
\text{Conclusion} &
\end{align*}
\]

Meaning: For proving the Conclusion, it suffices to prove all Premisses.

**Example**

\[
\begin{align*}
\text{andRight} & \quad \Gamma \Rightarrow \phi, \Delta \quad \Gamma \Rightarrow \psi, \Delta \\
& \quad \Gamma \Rightarrow \phi \land \psi, \Delta
\end{align*}
\]

Admissible to have no premisses (iff conclusion is valid, e.g., axiom)

A rule is **sound** (correct) iff the validity of its premisses implies the validity of its conclusion.
<table>
<thead>
<tr>
<th>main</th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>$\Gamma \Rightarrow \phi, \Delta$</td>
<td>$\Gamma, \phi \Rightarrow \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \neg \phi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \neg \phi, \Delta$</td>
</tr>
</tbody>
</table>
### ‘Propositional’ Sequent Calculus Rules

<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>$\Gamma \Rightarrow \phi, \Delta$</td>
<td>$\Gamma, \phi \Rightarrow \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \neg \phi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \neg \phi, \Delta$</td>
</tr>
<tr>
<td><strong>and</strong></td>
<td>$\Gamma, \phi, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi, \Delta$ $\Gamma \Rightarrow \psi, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \phi \land \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi \land \psi, \Delta$</td>
</tr>
</tbody>
</table>
### 'Propositional' Sequent Calculus Rules

<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 \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><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 \phi \land \psi, \Delta )</td>
</tr>
<tr>
<td><strong>or</strong></td>
<td>( \Gamma, \phi \implies \Delta ) ( \Gamma, \psi \implies \Delta )</td>
<td>( \Gamma \implies \phi, \psi, \Delta )</td>
</tr>
<tr>
<td></td>
<td>( \Gamma, \phi \lor \psi \implies \Delta )</td>
<td>( \Gamma \implies \phi \lor \psi, \Delta )</td>
</tr>
</tbody>
</table>
‘Propositional’ Sequent Calculus Rules

<table>
<thead>
<tr>
<th>main</th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>$\Gamma \Rightarrow \phi, \Delta$</td>
<td>$\Gamma \Rightarrow \neg \phi, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \neg \phi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \neg \phi, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \phi, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi, \Delta$ $\Gamma \Rightarrow \psi, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \phi \land \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi \land \psi, \Delta$</td>
</tr>
<tr>
<td>and</td>
<td>$\Gamma \Rightarrow \phi, \Delta$ $\Gamma, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi, \psi, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \phi \lor \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi \lor \psi, \Delta$</td>
</tr>
<tr>
<td>or</td>
<td>$\Gamma \Rightarrow \phi, \Delta$ $\Gamma, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi \Rightarrow \psi, \Delta$</td>
</tr>
<tr>
<td>imp</td>
<td>$\Gamma \Rightarrow \phi \Rightarrow \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi \Rightarrow \psi, \Delta$</td>
</tr>
</tbody>
</table>
### ‘Propositional’ Sequent Calculus Rules

<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 \phi, \Delta)</td>
<td>(\Gamma \Rightarrow \neg \phi, \Delta)</td>
</tr>
<tr>
<td></td>
<td>(\Gamma, \neg \phi \Rightarrow \Delta)</td>
<td>(\Gamma \Rightarrow \neg \phi, \Delta)</td>
</tr>
<tr>
<td>and</td>
<td>(\Gamma, \phi, \psi \Rightarrow \Delta)</td>
<td>(\Gamma \Rightarrow \phi, \Delta) (\Gamma \Rightarrow \psi, \Delta)</td>
</tr>
<tr>
<td></td>
<td>(\Gamma, \phi \land \psi \Rightarrow \Delta)</td>
<td>(\Gamma \Rightarrow \phi \land \psi, \Delta)</td>
</tr>
<tr>
<td>or</td>
<td>(\Gamma, \phi \Rightarrow \Delta) (\Gamma, \psi \Rightarrow \Delta)</td>
<td>(\Gamma \Rightarrow \phi, \psi, \Delta)</td>
</tr>
<tr>
<td></td>
<td>(\Gamma \Rightarrow \phi \lor \psi, \Delta)</td>
<td>(\Gamma \Rightarrow \phi \lor \psi, \Delta)</td>
</tr>
<tr>
<td>imp</td>
<td>(\Gamma \Rightarrow \phi, \Delta) (\Gamma, \psi \Rightarrow \Delta)</td>
<td>(\Gamma \Rightarrow \phi \Rightarrow \psi, \Delta)</td>
</tr>
<tr>
<td></td>
<td>(\Gamma \Rightarrow \phi \Rightarrow \psi, \Delta)</td>
<td>(\Gamma \Rightarrow \phi \Rightarrow \psi, \Delta)</td>
</tr>
<tr>
<td>close</td>
<td>(\Gamma, \phi \Rightarrow \phi, \Delta)</td>
<td>(\Gamma \Rightarrow \phi \Rightarrow \phi, \Delta)</td>
</tr>
</tbody>
</table>

SEFM: First-Order Logic
<table>
<thead>
<tr>
<th></th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>not</strong></td>
<td>( \Gamma \Rightarrow \phi, \Delta ) ( \Gamma, \neg \phi \Rightarrow \Delta )</td>
<td>( \Gamma, \phi \Rightarrow \Delta ) ( \Gamma \Rightarrow \neg \phi, \Delta )</td>
</tr>
<tr>
<td><strong>and</strong></td>
<td>( \Gamma, \phi, \psi \Rightarrow \Delta ) ( \Gamma, \phi \land \psi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \phi, \Delta ) ( \Gamma \Rightarrow \psi, \Delta ) ( \Gamma \Rightarrow \phi \land \psi, \Delta )</td>
</tr>
<tr>
<td><strong>or</strong></td>
<td>( \Gamma, \phi \Rightarrow \Delta ) ( \Gamma, \psi \Rightarrow \Delta ) ( \Gamma, \phi \lor \psi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \phi, \psi, \Delta ) ( \Gamma \Rightarrow \phi \lor \psi, \Delta )</td>
</tr>
<tr>
<td><strong>imp</strong></td>
<td>( \Gamma \Rightarrow \phi, \Delta ) ( \Gamma, \psi \Rightarrow \Delta ) ( \Gamma, \phi \rightarrow \psi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \phi \rightarrow \psi, \Delta ) ( \Gamma, \phi \Rightarrow \psi, \Delta )</td>
</tr>
<tr>
<td><strong>close</strong></td>
<td>( \Gamma, \phi \Rightarrow \phi, \Delta )</td>
<td>( \Gamma \Rightarrow \text{true, } \Delta )</td>
</tr>
</tbody>
</table>
### ‘Propositional’ Sequent Calculus Rules

<table>
<thead>
<tr>
<th>main</th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>$\Gamma \Rightarrow \phi, \Delta$</td>
<td>$\Gamma, \phi \Rightarrow \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \lnot \phi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \lnot \phi, \Delta$</td>
</tr>
<tr>
<td>and</td>
<td>$\Gamma, \phi, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi, \Delta$ $\Gamma \Rightarrow \psi, \Delta$</td>
</tr>
<tr>
<td>or</td>
<td>$\Gamma, \phi \Rightarrow \Delta$ $\Gamma, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi \lor \psi, \Delta$</td>
</tr>
<tr>
<td>imp</td>
<td>$\Gamma \Rightarrow \phi, \Delta$ $\Gamma, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi \Rightarrow \psi, \Delta$</td>
</tr>
</tbody>
</table>

| close | $\Gamma, \phi \Rightarrow \phi, \Delta$ | true | $\Gamma \Rightarrow \text{true}, \Delta$ | false | $\Gamma, \text{false} \Rightarrow \Delta$ |
Goal to prove: $G = \psi_1, \ldots, \psi_m \Rightarrow \phi_1, \ldots, \phi_n$

- find rule $\mathcal{R}$ whose conclusion matches $G$
- instantiate $\mathcal{R}$ such that its conclusion is identical to $G$
- apply that instantiation to all premisses of $\mathcal{R}$, resulting in new goals $G_1, \ldots, G_r$
- recursively find proofs for $G_1, \ldots, G_r$
- tree structure with goal as root
- close proof branch when rule without premiss encountered
Sequent Calculus Proofs

**Goal** to prove: \( \mathcal{G} = \psi_1, \ldots, \psi_m \Rightarrow \phi_1, \ldots, \phi_n \)

- find rule \( \mathcal{R} \) whose conclusion matches \( \mathcal{G} \)
- instantiate \( \mathcal{R} \) such that its conclusion is identical to \( \mathcal{G} \)
- apply that instantiation to all premisses of \( \mathcal{R} \), resulting in new goals \( \mathcal{G}_1, \ldots, \mathcal{G}_r \)
- recursively find proofs for \( \mathcal{G}_1, \ldots, \mathcal{G}_r \)
- tree structure with goal as root
- close proof branch when rule without premiss encountered

**Goal-directed proof search**

In KeY tool proof displayed as Java Swing tree
A Simple Proof

\[ \Rightarrow (p \land (p \rightarrow q)) \rightarrow q \]
A Simple Proof

\[ p \land (p \to q) \Rightarrow q \]

\[ \Rightarrow (p \land (p \to q)) \to q \]
A Simple Proof

\[
\begin{array}{c}
\hline
p, (p \rightarrow q) \Rightarrow q \\
\hline
p \land (p \rightarrow q) \Rightarrow q \\
\hline
\Rightarrow (p \land (p \rightarrow q)) \rightarrow q
\end{array}
\]
A Simple Proof

\[
\begin{align*}
p & \implies p, q & p, q & \implies q \\
p, (p \implies q) & \implies q \\
p \land (p \implies q) & \implies q \\
\implies (p \land (p \implies q)) & \implies q
\end{align*}
\]
A Simple Proof

A proof is closed iff all its branches are closed.
A Simple Proof

A proof is **closed** iff all its branches are closed

#### Demo

```latex
\[
\text{CLOSE} \quad \text{*} \quad \text{CLOSE} \\
\begin{array}{c}
p \Rightarrow p, q \\
p, q \Rightarrow q \\
p, (p \rightarrow q) \Rightarrow q \\
p \land (p \rightarrow q) \Rightarrow q \\
\Rightarrow (p \land (p \rightarrow q)) \rightarrow q
\end{array}
\]```

prop.key
Proving a universally quantified formula

Claim: $\forall \tau \, x; \phi$ is true

How is such a claim proved in mathematics?
Proving a universally quantified formula

Claim: \( \forall \tau \ x; \ \phi \) is true

How is such a claim proved in mathematics?

All even numbers are divisible by 2 \( \forall \text{int} \ x; \ (\text{even}(x) \rightarrow \text{divByTwo}(x)) \)
Proving a universally quantified formula

Claim: $\forall \tau \, x; \phi$ is true

How is such a claim proved in mathematics?

All even numbers are divisible by 2

$\forall \text{int} \, x; (\text{even}(x) \rightarrow \text{divByTwo}(x))$

Let $c$ be an arbitrary number

Declare “unused” constant $\text{int} \, c$
Proving a universally quantified formula

Claim: $\forall \tau x; \phi$ is true

How is such a claim proved in mathematics?

All even numbers are divisible by 2

Let $c$ be an arbitrary number

The even number $c$ is divisible by 2

\[ \forall \text{int } x; (\text{even}(x) \rightarrow \text{divByTwo}(x)) \]

Declare “unused” constant $\text{int } c$

prove $\text{even}(c) \rightarrow \text{divByTwo}(c)$
Proving Validity of First-Order Formulas

Proving a universally quantified formula

Claim: \( \forall \tau \; x ; \phi \) is true

How is such a claim proved in mathematics?

All even numbers are divisible by 2
\( \forall \text{int } x ; (\text{even}(x) \rightarrow \text{divByTwo}(x)) \)

Let \( c \) be an arbitrary number
Declare “unused” constant \( \text{int } c \)

The even number \( c \) is divisible by 2
prove \( \text{even}(c) \rightarrow \text{divByTwo}(c) \)

Sequent rule \( \forall \)-right

\[
\frac{\Gamma \Rightarrow [x/c] \phi, \Delta}{\Gamma \Rightarrow \forall \tau \; x ; \phi, \Delta}
\]

- \([x/c] \phi \) is result of replacing each occurrence of \( x \) in \( \phi \) with \( c \)
- \( c \) new constant of type \( \tau \)
Proving an existentially quantified formula

Claim: $\exists \tau x; \phi$ is true

How is such a claim proved in mathematics?
Proving an existentially quantified formula

Claim: $\exists \tau x; \phi$ is true

How is such a claim proved in mathematics?

There is at least one prime number $\exists \text{int } x; \text{prime}(x)$
Proving an existentially quantified formula

Claim: \( \exists \tau x; \phi \) is true

How is such a claim proved in mathematics?

There is at least one prime number \( \exists \text{int} \ x; \text{prime}(x) \)

Provide any “witness”, say, 7

Use variable-free term \( \text{int} \ 7 \)

\text{Sequent rule} \ \exists \text{-right} \\
\Gamma = \Rightarrow [x/t] \phi, \exists \tau x; \phi, \Delta \\ \Rightarrow \exists \tau x; \phi, \Delta 

\text{Proof might not work with } t! \text{ Need to keep premise to try again}
Proving an existentially quantified formula

Claim: $\exists \tau x; \phi$ is true

How is such a claim proved in mathematics?

There is at least one prime number

Provide any “witness”, say, 7

7 is a prime number

Use variable-free term $\text{int } 7$
Proving an existentially quantified formula

Claim: $\exists \tau \ x; \phi$ is true

How is such a claim proved in mathematics?

There is at least one prime number

$\exists \text{int} \ x; \ \text{prime}(x)$

Provide any "witness", say, 7

Use variable-free term $\text{int} \ 7$

7 is a prime number

prime(7)

Sequent rule $\exists$-right

$\Gamma \Rightarrow \exists \tau \ x; \phi, \Delta$

$\Gamma \Rightarrow \exists \tau \ x; \phi, \Delta$

$t$ any variable-free term of type $\tau$

Proof might not work with $t$! Need to keep premise to try again
Using a universally quantified formula

We assume \( \forall \tau x; \phi \) is true

How is such a fact used in a mathematical proof?
Using a universally quantified formula

We assume $\forall \tau x; \phi$ is true

How is such a fact *used* in a mathematical proof?

We know that all primes are odd $\forall \text{int } x; (\text{prime}(x) \rightarrow \text{odd}(x))$
Using a universally quantified formula

We assume $\forall \tau x; \phi$ is true

How is such a fact used in a mathematical proof?

We know that all primes are odd $\forall \text{int } x; \text{prime}(x) \rightarrow \text{odd}(x))$

In particular, this holds for 17 Use variable-free term $\text{int} \ 17$
Using a universally quantified formula

We assume $\forall \tau x; \phi$ is true

How is such a fact used in a mathematical proof?

We know that all primes are odd

$\forall \text{int } x; (\text{prime}(x) \rightarrow \text{odd}(x))$

In particular, this holds for 17

Use variable-free term $\text{int } 17$

We know: if 17 is prime it is odd

$\text{prime}(17) \rightarrow \text{odd}(17)$
Using a universally quantified formula

We assume $\forall \tau x; \phi$ is true

How is such a fact used in a mathematical proof?

We know that all primes are odd

$\forall \text{int} x; (\text{prime}(x) \rightarrow \text{odd}(x))$

In particular, this holds for 17

Use variable-free term $\text{int} \ 17$

We know: if 17 is prime it is odd

$\text{prime}(17) \rightarrow \text{odd}(17)$

Sequent rule $\forall$-left

$$\frac{\Gamma, \forall \tau x; \phi, [x/t'] \phi \Rightarrow \Delta}{\Gamma, \forall \tau x; \phi \Rightarrow \Delta}$$

- $t'$ any variable-free term of type $\tau$
- We might need other instances besides $t'$! Keep premise $\forall \tau x; \phi$
Using an existentially quantified formula

We assume \( \exists \tau \ x; \ \phi \) is true

How is such a fact used in a mathematical proof?
Using an existentially quantified formula

We assume $\exists \tau x; \phi$ is true

How is such a fact used in a mathematical proof?

We know such an element exists. Let's give it a new name for future reference.
Using an existentially quantified formula

We assume $\exists \tau x; \phi$ is true

How is such a fact used in a mathematical proof?

We know such an element exists. Let’s give it a new name for future reference.

Sequent rule $\exists$-left

\[
\text{existsLeft} \quad \frac{\Gamma, [x/c] \phi \Rightarrow \Delta}{\Gamma, \exists \tau x; \phi \Rightarrow \Delta}
\]

- $c$ new constant of type $\tau$
Using an existentially quantified formula
Using an existentially quantified formula

Let \( x, y \) denote integer constants, both are not zero.

\[ \neg(x = 0), \neg(y = 0) \]
Using an existentially quantified formula

Let $x, y$ denote integer constants, both are not zero. We know further that $x$ divides $y$.

\[ \neg(x = 0), \neg(y = 0), \exists \text{int } k; k \times x = y \implies \]
Using an existentially quantified formula

Let $x, y$ denote integer constants, both are not zero. We know further that $x$ divides $y$.

**Show:** $(y/x) \times x = y$ (’/’ is division on integers, i.e. the equation is not always true, e.g. $x = 2, y = 1$)

$(y/x) \times x = y \iff \neg(x = 0), \neg(y = 0), \exists \text{int } k; k \times x = y \Rightarrow (y/x) \times x = y$
Using an existentially quantified formula

Let $x, y$ denote integer constants, both are not zero. We know further that $x$ divides $y$.

**Show:** $(y/x) \times x = y$ (′/′ is division on integers, i.e. the equation is not always true, e.g. $x = 2, y = 1$)

**Proof:** We know $x$ divides $y$, i.e. there exists a $k$ such that $k \times x = y$. Let now $c$ denote such a $k$.

\[
\neg(x = 0), \neg(y = 0), \exists \text{int } k; k \times x = y \Rightarrow (y/x) \times x = y
\]
Using an existentially quantified formula

Let $x, y$ denote integer constants, both are not zero. We know further that $x$ divides $y$.

**Show:** $(y/x) \times x = y$ (’/’ is division on integers, i.e. the equation is not always true, e.g. $x = 2, y = 1$)

**Proof:** We know $x$ divides $y$, i.e. there exists a $k$ such that $k \times x = y$. Let now $c$ denote such a $k$. Hence we can replace $y$ by $c \times x$ on the right side (see slide 35).

\[
\neg(x = 0), \neg(y = 0), c \times x = y \implies ((c \times x)/x) \times x = y
\]

\[
\neg(x = 0), \neg(y = 0), c \times x = y \implies (y/x) \times x = y
\]

\[
\neg(x = 0), \neg(y = 0), \exists \text{ int } k; k \times x = y \implies (y/x) \times x = y
\]
Using an existentially quantified formula

Let $x, y$ denote integer constants, both are not zero. We know further that $x$ divides $y$.

**Show:** $(y/x) \times x = y$ ('/' is division on integers, i.e. the equation is not always true, e.g. $x = 2, y = 1$)

**Proof:** We know $x$ divides $y$, i.e. there exists a $k$ such that $k \times x = y$. Let now $c$ denote such a $k$. Hence we can replace $y$ by $c \times x$ on the right side (see slide 35). ... □

\[
\begin{align*}
\neg(x = 0), \neg(y = 0), c \times x = y & \Rightarrow ((c \times x)/x) \times x = y \\
\neg(x = 0), \neg(y = 0), c \times x = y & \Rightarrow (y/x) \times x = y \\
\neg(x = 0), \neg(y = 0), \exists \text{ int } k; k \times x = y & \Rightarrow (y/x) \times x = y
\end{align*}
\]
Example (A simple theorem about binary relations)

\[ \exists x; \forall y; p(x, y) \Rightarrow \forall y; \exists x; p(x, y) \]

Untyped logic: let static type of \( x \) and \( y \) be \( \top \)
Example (A simple theorem about binary relations)

\[
\forall y; p(c, y) \Rightarrow \forall y; \exists x; p(x, y) \\
\exists x; \forall y; p(x, y) \Rightarrow \forall y; \exists x; p(x, y)
\]

\[\exists\text{-left: substitute new constant } c \text{ of type } \top \text{ for } x\]
Example (A simple theorem about binary relations)

\[
\begin{align*}
\forall y; p(c, y) & \implies \exists x; p(x, d) \\
\therefore \forall y; p(c, y) & \implies \forall y; \exists x; p(x, y) \\
\exists x; \forall y; p(x, y) & \implies \forall y; \exists x; p(x, y)
\end{align*}
\]

\(\forall\)-right: substitute new constant \(d\) of type \(\top\) for \(y\)
Example (A simple theorem about binary relations)

\[
\begin{align*}
p(c, d), \forall y; \ p(c, y) & \implies \exists x; \ p(x, d) \\
\forall y; \ p(c, y) & \implies \exists x; \ p(x, d) \\
\forall y; \ p(c, y) & \implies \forall y; \ \exists x; \ p(x, y) \\
\exists x; \ \forall y; \ p(x, y) & \implies \forall y; \ \exists x; \ p(x, y)
\end{align*}
\]

\textit{-left: free to substitute any term of type } \top \textit{ for } y \textit{, choose } d
Example (A simple theorem about binary relations)

\[
\begin{align*}
p(c, d), \forall y; p(c, y) &\Rightarrow p(c, d), \exists x; p(x, y) \\
p(c, d), \forall y; p(c, y) &\Rightarrow \exists x; p(x, d) \\
\forall y; p(c, y) &\Rightarrow \exists x; p(x, d) \\
\forall y; p(c, y) &\Rightarrow \forall y; \exists x; p(x, y) \\
\exists x; \forall y; p(x, y) &\Rightarrow \forall y; \exists x; p(x, y)
\end{align*}
\]

\(\exists\)-right: free to substitute any term of type \(\top\) for \(x\), choose \(c\)
Example (A simple theorem about binary relations)

\[ p(c, d), \forall y; p(c, y) \Rightarrow p(c, d), \exists x; p(x, y) \]

\[ p(c, d), \forall y; p(c, y) \Rightarrow \exists x; p(x, d) \]

\[ \forall y; p(c, y) \Rightarrow \exists x; p(x, d) \]

\[ \forall y; p(c, y) \Rightarrow \forall y; \exists x; p(x, y) \]

\[ \exists x; \forall y; p(x, y) \Rightarrow \forall y; \exists x; p(x, y) \]

Close
Proving Validity of First-Order Formulas Cont’d

Example (A simple theorem about binary relations)

\[
\begin{align*}
\ast & \quad p(c, d), \forall y; p(c, y) \implies p(c, d), \exists x; p(x, y) \\
p(c, d), \forall y; p(c, y) & \implies \exists x; p(x, d) \\
\forall y; p(c, y) & \implies \exists x; p(x, d) \\
\forall y; p(c, y) & \implies \forall y; \exists x; p(x, y) \\
\exists x; \forall y; p(x, y) & \implies \forall y; \exists x; p(x, y)
\end{align*}
\]

Demo

relSimple.key
**Using an equation between terms**

We assume $t = t'$ is true

How is such a fact used in a mathematical proof?
Using an equation between terms

We assume \( t = t' \) is true

How is such a fact used in a mathematical proof?

Use \( x = y - 1 \) to simplify \( x + 1/y \)

\[
x = y - 1 \implies 1 = x + 1/y
\]
Using an equation between terms

We assume $t = t'$ is true

How is such a fact used in a mathematical proof?

Use $x = y - 1$ to simplify $x + 1/y$

$x = y - 1 \implies 1 = x + 1/y$

Replace $x$ in conclusion with right-hand side of equation
We assume \( t = t' \) is true

How is such a fact used in a mathematical proof?

Use \( x = y - 1 \) to simplify \( x + 1/y \)

\[
x = y - 1 \implies 1 = x + 1/y
\]

Replace \( x \) in conclusion with right-hand side of equation

We know: \( x + 1/y \) equal to \( y - 1 + 1/y \)

\[
x = y - 1 \implies 1 = y - 1 + 1/y
\]
Using an equation between terms

We assume $t = t'$ is true

How is such a fact used in a mathematical proof?

Use $x = y - 1$ to simplify $x + 1/y$

$x = y - 1 \Rightarrow 1 = x + 1/y$

Replace $x$ in conclusion with right-hand side of equation

We know: $x + 1/y$ equal to $y - 1 + 1/y$

$x = y - 1 \Rightarrow 1 = y - 1 + 1/y$

Sequent rule $\Rightarrow$-left

applyEqL  \[
\frac{\Gamma, t = t', [t/t'] \phi \Rightarrow \Delta}{\Gamma, t = t', \phi \Rightarrow \Delta}
\]

applyEqR  \[
\frac{\Gamma, t = t' \Rightarrow [t/t'] \phi, \Delta}{\Gamma, t = t' \Rightarrow \phi, \Delta}
\]

- Always replace left- with right-hand side (use eqSymm if necessary)
- $t, t'$ variable-free terms of the same type
Closing a subgoal in a proof

- We derived a sequent that is obviously valid

\[
\begin{align*}
\text{close} & \quad \Gamma, \phi \Rightarrow \phi, \Delta \\
\text{true} & \quad \Gamma \Rightarrow \text{true}, \Delta \\
\text{false} & \quad \Gamma, \text{false} \Rightarrow \Delta 
\end{align*}
\]

- We derived an equation that is obviously valid

\[
\begin{align*}
\text{eqClose} & \quad \Gamma \Rightarrow t = t, \Delta 
\end{align*}
\]
### Sequent Calculus for FOL at One Glance

<table>
<thead>
<tr>
<th>left side, antecedent</th>
<th>right side, succedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall ) ( \Gamma, \forall \tau x; \phi, [x/t]\phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow [x/c]\phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, \forall \tau x; \phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \forall \tau x; \phi, \Delta )</td>
</tr>
<tr>
<td>( \exists ) ( \Gamma, [x/c]\phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow [x/t]\phi, \exists \tau x; \phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, \exists \tau x; \phi \Rightarrow \Delta )</td>
<td>( \Gamma \Rightarrow \exists \tau x; \phi, \Delta )</td>
</tr>
<tr>
<td>( = ) ( \Gamma, t = t' \Rightarrow [t/t']\phi, \Delta )</td>
<td>( \Gamma \Rightarrow t = t, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, t = t' \Rightarrow \phi, \Delta )</td>
<td></td>
</tr>
<tr>
<td>(+ application rule on left side)</td>
<td></td>
</tr>
</tbody>
</table>

- \([t/t']\phi\) is result of replacing each occurrence of \(t\) in \(\phi\) with \(t'\)
- \(t,t'\) variable-free terms of type \(\tau\)
- \(c\) **new** constant of type \(\tau\) (occurs not on current proof branch)
- Equations can be reversed by commutativity
Recap: ‘Propositional’ Sequent Calculus Rules

<table>
<thead>
<tr>
<th>main</th>
<th>left side (antecedent)</th>
<th>right side (succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>imp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>close</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**left side (antecedent)**

- **not**
  
  \[ \Gamma \Rightarrow \phi, \Delta \]
  \[ \Gamma, \neg \phi \Rightarrow \Delta \]

- **and**
  
  \[ \Gamma, \phi, \psi \Rightarrow \Delta \]
  \[ \Gamma, \phi \land \psi \Rightarrow \Delta \]

- **or**
  
  \[ \Gamma, \phi \Rightarrow \Delta \]
  \[ \Gamma, \psi \Rightarrow \Delta \]
  \[ \Gamma, \phi \lor \psi \Rightarrow \Delta \]

- **imp**
  
  \[ \Gamma \Rightarrow \phi, \Delta \]
  \[ \Gamma, \psi \Rightarrow \Delta \]
  \[ \Gamma, \phi \Rightarrow \psi \Rightarrow \Delta \]
  \[ \Gamma, \phi \rightarrow \psi \Rightarrow \Delta \]

**right side (succedent)**

- **not**
  
  \[ \Gamma, \phi \Rightarrow \Delta \]
  \[ \Gamma \Rightarrow \neg \phi, \Delta \]

- **and**
  
  \[ \Gamma \Rightarrow \phi, \Delta \]
  \[ \Gamma \Rightarrow \psi, \Delta \]
  \[ \Gamma \Rightarrow \phi \land \psi, \Delta \]

- **or**
  
  \[ \Gamma \Rightarrow \phi, \psi, \Delta \]
  \[ \Gamma \Rightarrow \phi \lor \psi, \Delta \]

- **imp**
  
  \[ \Gamma \Rightarrow \phi \rightarrow \psi, \Delta \]

- **close**
  
  \[ \Gamma, \phi \Rightarrow \phi, \Delta \]

- **true**
  
  \[ \Gamma \Rightarrow \text{true}, \Delta \]

- **false**
  
  \[ \Gamma, \text{false} \Rightarrow \Delta \]
### Feature List

- Can work on multiple proofs simultaneously (task list)
- Proof trees visualized as **JAVA** Swing tree
- Point-and-click navigation within proof
- Undo proof steps, prune proof trees
- Pop-up menu with proof rules applicable in pointer focus
- Preview of rule effect as tool tip
- Quantifier instantiation and equality rules by drag-and-drop
- Possible to hide (and unhide) parts of a sequent
- Saving and loading of proofs

**Demo**

rel.key, twoInstances.key
Literature for this Lecture

essential:

▶ W. Ahrendt
  Using KeY
  Chapter 10 in [KeYbook]

further reading:

▶ M. Giese
  First-Order Logic
  Chapter 2 in [KeYbook]

Part II

First-Order Semantics
First-Order Semantics

From propositional to first-order semantics

- In prop. logic, an interpretation of variables with \{T, F\} sufficed
- In first-order logic we must assign meaning to:
  - variables bound in quantifiers
  - constant and function symbols
  - predicate symbols
- Each variable or function value may denote a different item
- Respect typing: \texttt{int i, List l} \textbf{must} denote different items

What we need (to interpret a first-order formula)

1. A collection of \textbf{typed universes} of items
2. A mapping from \textbf{variables} to items
3. A mapping from \textbf{function} arguments to function values
4. The set of argument tuples where a \textbf{predicate} is true
First-Order Domains/Universes

1. A collection of typed universes of items

**Definition (Universe/Domain)**

A non-empty set $\mathcal{D}$ of items is a universe or domain

Each element of $\mathcal{D}$ has a fixed type given by $\delta : \mathcal{D} \rightarrow \tau$

- Notation for the domain elements of type $\tau \in \mathcal{T}$:
  \[ \mathcal{D}^\tau = \{ d \in \mathcal{D} \mid \delta(d) = \tau \} \]

- Each type $\tau \in \mathcal{T}$ must ‘contain’ at least one domain element:
  \[ \mathcal{D}^\tau \neq \emptyset \]
First-Order States

3. A mapping from function arguments to function values
4. The set of argument tuples where a predicate is true

Definition (First-Order State)
Let $\mathcal{D}$ be a domain with typing function $\delta$
Let $f$ be declared as $\tau \ f(\tau_1, \ldots, \tau_r)$;
Let $p$ be declared as $p(\tau_1, \ldots, \tau_r)$;
Let $\mathcal{I}(f) : \mathcal{D}^{\tau_1} \times \cdots \times \mathcal{D}^{\tau_r} \to \mathcal{D}^\tau$
Let $\mathcal{I}(p) \subseteq \mathcal{D}^{\tau_1} \times \cdots \times \mathcal{D}^{\tau_r}$

Then $S = (\mathcal{D}, \delta, \mathcal{I})$ is a first-order state
First-Order States Cont’d

Example

Signature: int i; short j; int f(int); Object obj; <(int,int);
\(\mathcal{D} = \{17, 2, o\}\) where all numbers are short

\[
\begin{align*}
\mathcal{I}(i) &= 17 \\
\mathcal{I}(j) &= 17 \\
\mathcal{I}(\text{obj}) &= o
\end{align*}
\]

<table>
<thead>
<tr>
<th>(\mathcal{D}^{\text{int}})</th>
<th>(\mathcal{I}(f))</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c}
\mathcal{D}^{\text{int}} \times \mathcal{D}^{\text{int}} & \text{in } \mathcal{I}(<) \? \\
\hline
(2, 2) & F \\
(2, 17) & T \\
(17, 2) & F \\
(17, 17) & F \\
\end{array}
\]

One of uncountably many possible first-order states!
Semantics of Reserved Signature Symbols

Definition

Equality symbol $=\,$ declared as $=\,(\top, \top)$

Interpretation is fixed as $\mathcal{I}(=) = \{(d, d) \mid d \in \mathcal{D}\}$

“Referential Equality” (holds if arguments refer to identical item)

Exercise: write down the predicate table for example domain
Signature Symbols vs. Domain Elements

- Domain elements different from the terms representing them
- First-order formulas and terms have no access to domain

**Example**

Signature: Object obj1, obj2;
Domain: $\mathcal{D} = \{o\}$

In this state, necessarily $\mathcal{I}(\text{obj1}) = \mathcal{I}(\text{obj2}) = o$
2. A mapping from variables to objects

Think of variable assignment as environment for storage of local variables

**Definition (Variable Assignment)**

A variable assignment $\beta$ maps variables to domain elements. It respects the variable type, i.e., if $x$ has type $\tau$ then $\beta(x) \in D^\tau$.

**Definition (Modified Variable Assignment)**

Let $y$ be variable of type $\tau$, $\beta$ variable assignment, $d \in D^\tau$:

$$\beta^d_y(x) := \begin{cases} 
\beta(x) & x \neq y \\
 d & x = y 
\end{cases}$$
Given a first-order state $S$ and a variable assignment $\beta$, it is possible to evaluate first-order terms under $S$ and $\beta$.

**Definition (Valuation of Terms)**

$val_{S,\beta} : \text{Term} \rightarrow \mathcal{D}$ such that $val_{S,\beta}(t) \in \mathcal{D}^\tau$ for $t \in \text{Term}_\tau$:

- $val_{S,\beta}(x) = \beta(x)$
- $val_{S,\beta}(f(t_1, \ldots, t_r)) = I(f)(val_{S,\beta}(t_1), \ldots, val_{S,\beta}(t_r))$
Semantic Evaluation of Terms Cont’d

Example

Signature: int i; short j; int f(int);
\( D = \{17, 2, o\} \) where all numbers are short
Variables: Object obj; int x;

\[
\begin{array}{c}
I(i) = 17 \\
I(j) = 17
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
D & I(f) \\
\hline
2 & 17 \\
17 & 2 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
Var & \beta \\
\hline
obj & o \\
x & 17 \\
\hline
\end{array}
\]

- \( \text{val}_{S,\beta}(f(f(i))) \)?
- \( \text{val}_{S,\beta}(x) \)?
Definition (Valuation of Formulas)

\[ \text{val}_{S, \beta}(\phi) \text{ for } \phi \in \text{For} \]

- \[ \text{val}_{S, \beta}(p(t_1, \ldots, t_r)) = T \iff (\text{val}_{S, \beta}(t_1), \ldots, \text{val}_{S, \beta}(t_r)) \in I(p) \]
- \[ \text{val}_{S, \beta}(\phi \land \psi) = T \iff \text{val}_{S, \beta}(\phi) = T \text{ and } \text{val}_{S, \beta}(\psi) = T \]
- 
- \[ \ldots \text{as in propositional logic} \]

- \[ \text{val}_{S, \beta}(\forall \tau x; \phi) = T \iff \text{val}_{S, \beta^d}(\forall \tau x; \phi) = T \text{ for all } d \in D^\tau \]
- \[ \text{val}_{S, \beta}(\forall \tau x; \phi) = T \iff \text{val}_{S, \beta^d}(\forall \tau x; \phi) = T \text{ for at least one } d \in D^\tau \]
Semantic Evaluation of Formulas Cont’d

Example

Signature: short j; int f(int); Object obj; <(int,int);
\( \mathcal{D} = \{17, 2, o\} \) where all numbers are short

\[\begin{align*}
\mathcal{I}(j) &= 17 \\
\mathcal{I}(\text{obj}) &= o
\end{align*}\]

\[
\begin{array}{c|c}
\mathcal{D}_{\text{int}} & \mathcal{I}(f) \\
\hline
2 & 2 \\
17 & 2 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\mathcal{D}_{\text{int}} \times \mathcal{D}_{\text{int}} & \text{in } \mathcal{I}(<)? \\
\hline
(2, 2) & F \\
(2, 17) & T \\
(17, 2) & F \\
(17, 17) & F \\
\end{array}
\]

- \(\text{val}_{S,\beta}(f(j) < j)\) ?
- \(\text{val}_{S,\beta}(\exists \text{int } x; f(x) = x)\) ?
- \(\text{val}_{S,\beta}(\forall \text{Object } o1; \forall \text{Object } o2; o1 = o2)\) ?
Semantic Notions

Definition (Satisfiability, Truth, Validity)

\[ \text{val}_{S,\beta}(\phi) = T \]
\[ S \models \phi \quad \text{iff} \quad \text{for all } \beta : \text{val}_{S,\beta}(\phi) = T \]
\[ \models \phi \quad \text{iff} \quad \text{for all } S : \quad S \models \phi \]

(\(\phi\) is satisfiable)
(\(\phi\) is true in \(S\))
(\(\phi\) is valid)

Closed formulas that are satisfiable are also true: one top-level notion
## Semantic Notions

### Definition (Satisfiability, Truth, Validity)

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$val_{S,\beta}(\phi) = T$</td>
<td>($\phi$ is satisfiable)</td>
</tr>
<tr>
<td>$S \models \phi$</td>
<td>iff for all $\beta : val_{S,\beta}(\phi) = T$ ($\phi$ is true in $S$)</td>
</tr>
<tr>
<td>$\models \phi$</td>
<td>iff for all $S : S \models \phi$ ($\phi$ is valid)</td>
</tr>
</tbody>
</table>

Closed formulas that are satisfiable are also true: one top-level notion

### Example

- $f(j) < j$ is true in $S$
- $\exists \text{int } x; \ i = x$ is valid
- $\exists \text{int } x; \neg(x = x)$ is not satisfiable