Install the KeY-Tool...

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

▷ We recommend using Java Web Start
▷ Alternatively, install KeY locally, as binary or Eclipse plugin
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
- Sequent calculus for proving FOL formulas
- KeY system as propositional, and first-order, prover (for now)
- Formal semantics
Part I

The Language of FOL
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 $\Sigma_1 +$ Constants**

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

\[\begin{align*}
\alpha_{\Sigma_1}(<) &= (\text{int},\text{int}) \\
\alpha_{\Sigma_1}(+) &= \alpha_{\Sigma_1}(-) &= (\text{int},\text{int},\text{int}) \\
\alpha_{\Sigma_1}(0) &= \alpha_{\Sigma_1}(1) &= \alpha_{\Sigma_1}(-1) &= \ldots &= (\text{int})
\end{align*}\]

**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: $\ldots,-2,-1,0,1,2,\ldots$
Syntax of First-Order Logic: Signature Cont’d

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 $f()$.

Example

<table>
<thead>
<tr>
<th>Variables</th>
<th>integerArray a; int i;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicate Symbols</td>
<td>isEmpty(List); alertOn;</td>
</tr>
<tr>
<td>Function Symbols</td>
<td>int arrayLookup(int); Object o;</td>
</tr>
</tbody>
</table>
Typing of Signature:

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

can alternatively be written as:

\(<(\text{int}, \text{int});
\text{int } + (\text{int}, \text{int});
\text{int } 0; \text{ int } 1; \text{ int } -1; \ldots \)
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 $\Sigma_1$

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

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

Our variant of FOL allows infix notation for common functions:

- $-2 + 99$
- $7 - 8$
- $(7 - 8) + 1$
- $(v_1 - 8) + v_2$
Atomic Formulas

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 $p$. 
Atomic Formulas over Signature $\Sigma_1$

Example formulas over $\Sigma_1$: 
(assume variable int $v$;)

- $7 = 8$
- $< (7, 8)$
- $< (-2 - v, 99)$
- $< (v, v + 1)$

Our variant of FOL allows infix notation for common predicates:

- $7 < 8$
- $-2 - v < 99$
- $v < v + 1$
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’.
First-order Formulas: Examples

(signatures/types left out here)

Example (There are at least two elements)
\[ \exists x, y; \neg(x = y) \]

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?)
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:

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

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

The following formulas are valid:

1. $\neg(\exists \tau \ x; \ \phi) \iff \forall \tau \ x; \ \neg \phi$
2. $\neg(\forall \tau \ x; \ \phi) \iff \exists \tau \ x; \ \neg \phi$
3. $(\forall \tau \ x; \phi \land \psi) \iff (\forall \tau \ x; \phi) \land (\forall \tau \ x; \psi)$
4. $(\exists \tau \ x; \phi \lor \psi) \iff (\exists \tau \ x; \phi) \lor (\exists \tau \ x; \psi)$

Are the following formulas also valid?

5. $(\forall \tau \ x; \phi \lor \psi) \iff (\forall \tau \ x; \phi) \lor (\forall \tau \ x; \psi)$
6. $(\exists \tau \ x; \phi \land \psi) \iff (\exists \tau \ x; \phi) \land (\exists \tau \ x; \psi)$
## 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>( \land )</td>
<td>&amp;&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>Disjunction</td>
<td>( \lor )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implication</td>
<td>( \to, \supset )</td>
<td>( \rightarrow )</td>
<td>( \rightarrow )</td>
</tr>
<tr>
<td>Equivalence</td>
<td>( \iff )</td>
<td>( \leftrightarrow )</td>
<td>( \leftrightarrow )</td>
</tr>
<tr>
<td>Universal Quantifier</td>
<td>( \forall x; \phi )</td>
<td>n/a</td>
<td>( \forall \tau x; \phi )</td>
</tr>
<tr>
<td>Existential Quantifier</td>
<td>( \exists x; \phi )</td>
<td>n/a</td>
<td>( \exists \tau x; \phi )</td>
</tr>
<tr>
<td>Value equality</td>
<td>=</td>
<td>==</td>
<td>=</td>
</tr>
</tbody>
</table>
Part II

Sequent Calculus for FOL
Motivation for a Sequent Calculus

How to show a formula valid in propositional logic?
→ use a semantic truth table.

How about FOL? Formula: \( \text{isEven}(x) \lor \text{isOdd}(x) \)

<table>
<thead>
<tr>
<th>( x )</th>
<th>( \text{isEven}(x) )</th>
<th>( \text{isOdd}(x) )</th>
<th>( \text{isEven}(x) \lor \text{isOdd}(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( F )</td>
<td>( T )</td>
<td>( T )</td>
</tr>
<tr>
<td>2</td>
<td>( T )</td>
<td>( F )</td>
<td>( T )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Checking validity via **semantics** does not work.

Instead...
Reasoning by Syntactic Transformation

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

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

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

Antecedent

Succedent

has same meaning as

$$(\psi_1 \land \cdots \land \psi_m) \implies (\phi_1 \lor \cdots \lor \phi_n)$$

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

**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

Here, we call \( \phi \land \psi \) main formula and \( \Gamma, \Delta \) side formulas of sequent
Sequent Calculus Rules

Write syntactic transformation schema for sequents that reflects semantics of connectives

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

Premisses

RuleName

Conclusion

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)

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

<table>
<thead>
<tr>
<th>Left Side (Antecedent)</th>
<th>Right Side (Succedent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash \phi, \Delta )</td>
<td>( \Gamma, \phi \vdash \Delta )</td>
</tr>
<tr>
<td>( \Gamma \vdash \phi, \Delta )</td>
<td>( \Gamma \vdash \neg \phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, \phi \land \psi \vdash \Delta )</td>
<td>( \Gamma \vdash \phi, \Delta ) ( \Gamma \vdash \psi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, \phi \vdash \Delta ) ( \Gamma, \psi \vdash \Delta )</td>
<td>( \Gamma \vdash \phi \land \psi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma \vdash \phi, \Delta ) ( \Gamma \vdash \psi, \Delta )</td>
<td>( \Gamma \vdash \phi \lor \psi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma \vdash \phi \lor \psi, \Delta )</td>
<td>( \Gamma \vdash \phi \rightarrow \psi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma \vdash \phi \rightarrow \psi, \Delta )</td>
<td>( \Gamma \vdash \phi \rightarrow \psi, \Delta )</td>
</tr>
</tbody>
</table>

**Rules:***
- **Close:** \( \Gamma, \phi \Rightarrow \phi, \Delta \)
- **True:** \( \Gamma \Rightarrow \text{true}, \Delta \)
- **False:** \( \Gamma, \text{false} \Rightarrow \Delta \)

**Sequent Symbols:**
- \( \Gamma \): left side (antecedent)
- \( \Delta \): right side (succedent)
- \( \phi \): formula
- \( \neg \phi \): negation of \( \phi \)
- \( \phi \land \psi \): conjunction of \( \phi \) and \( \psi \)
- \( \phi \lor \psi \): disjunction of \( \phi \) and \( \psi \)
- \( \phi \rightarrow \psi \): implication from \( \phi \) to \( \psi \)
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

A proof is closed iff all its branches are closed

Demo

prop.key
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

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

- \([x/c] \phi\) 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?

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 $\text{prime}(7)$

Sequent rule $\exists$-right

\[
\begin{array}{c}
\text{existsRight} \\
\Gamma \Rightarrow [x/t] \phi, \exists \tau x; \phi, \Delta \\
\hline
\Gamma \Rightarrow \exists \tau x; \phi, \Delta
\end{array}
\]

- $t$ any variable-free term of type $\tau$
- Proof might not work with $t$! Need to keep premise to try again
Proving Validity of First-Order Formulas Cont’d

**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**

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

- $t'$ 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?

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

Sequent rule $\exists$-left

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

$\triangleright c$ new constant of type $\tau$
Proving Validity of First-Order Formulas Cont’d

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

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

**Sequent rule \( \Gamma \equiv \Delta \)**

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

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

- 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

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

- We derived an equation that is obviously valid

\[
\text{eqClose} : \Gamma \Rightarrow t = t, \Delta
\]
## Sequent Calculus for FOL at One Glance

<table>
<thead>
<tr>
<th></th>
<th>left side, antecedent</th>
<th>right side, succedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\forall)</td>
<td>(\Gamma, \forall \tau x; \phi, [x/t'] \phi \implies \Delta)</td>
<td>(\Gamma \implies [x/c] \phi, \Delta)</td>
</tr>
<tr>
<td></td>
<td>(\Gamma, \forall \tau x; \phi \implies \Delta)</td>
<td>(\Gamma \implies \forall \tau x; \phi, \Delta)</td>
</tr>
<tr>
<td>(\exists)</td>
<td>(\Gamma, [x/c] \phi \implies \Delta)</td>
<td>(\Gamma \implies [x/t'] \phi, \exists \tau x; \phi, \Delta)</td>
</tr>
<tr>
<td></td>
<td>(\Gamma, \exists \tau x; \phi \implies \Delta)</td>
<td>(\Gamma \implies \exists \tau x; \phi, \Delta)</td>
</tr>
<tr>
<td>(\equiv)</td>
<td>(\Gamma, t = t' \implies [t/t'] \phi, \Delta)</td>
<td>(\Gamma \implies t = t, \Delta)</td>
</tr>
<tr>
<td></td>
<td>(\Gamma, t = t' \implies \phi, \Delta)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>((+ \text{ 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>$\Gamma, \phi, \Delta$</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>and</td>
<td>$\Gamma, \phi, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \phi \wedge \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \psi, \Delta$</td>
</tr>
<tr>
<td>or</td>
<td>$\Gamma, \phi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi, \psi, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi \vee \psi, \Delta$</td>
</tr>
<tr>
<td>imp</td>
<td>$\Gamma \Rightarrow \phi, \Delta$</td>
<td>$\Gamma \Rightarrow \phi \Rightarrow \psi, \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \psi \Rightarrow \Delta$</td>
<td>$\Gamma \Rightarrow \phi \Rightarrow \psi, \Delta$</td>
</tr>
<tr>
<td>close</td>
<td>$\Gamma, \phi \Rightarrow \phi, \Delta$</td>
<td>$\Gamma \Rightarrow \text{true}, \Delta$</td>
</tr>
<tr>
<td>true</td>
<td>$\Gamma \Rightarrow \text{true}, \Delta$</td>
<td>$\Gamma \Rightarrow \text{false} \Rightarrow \Delta$</td>
</tr>
<tr>
<td>false</td>
<td>$\Gamma, \text{false} \Rightarrow \Delta$</td>
<td></td>
</tr>
</tbody>
</table>
**Example (A simple theorem about binary relations)**

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

Untyped logic: let static type of \(x\) and \(y\) be \(\top\)

\(\exists\)-left: substitute **new** constant \(c\) of type \(\top\) for \(x\)

\(\forall\)-right: substitute **new** constant \(d\) of type \(\top\) for \(y\)

\(\forall\)-left: free to substitute any term of type \(\top\) for \(y\), choose \(d\)

\(\exists\)-right: free to substitute any term of type \(\top\) for \(x\), choose \(c\)

Close
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 \( y = k \times x \). Let now \( c \) denote such a \( k \). Hence we can replace \( y \) by \( c \times x \) on the right side. ... □

\[
\begin{align*}
\neg(x = 0), \neg(y = 0), y = c \times x & \implies (c \times x)/x \times x = y \\
\neg(x = 0), \neg(y = 0), y = c \times x & \implies (y/x) \times x = y \\
\neg(x = 0), \neg(y = 0), \exists \text{int } k; y = k \times x & \implies (y/x) \times x = y
\end{align*}
\]
Features of the KeY Theorem Prover

Demo

rel.key, twoInstances.key

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
Literature for this Lecture


- A more up-to-date version:
  W. Ahrendt, S. Grebing, *Using the KeY Prover* to appear in the new KeY Book, end 2016 (available via Google group or personal request)

further reading:
Part III

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:
  - function symbols
  - predicate symbols
  - variables bound in quantifiers
- Respect typing: int i, List l must denote different items

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

1. A typed domain of items
2. A mapping from function symbols to functions on items
3. A mapping from predicate symbols to relation on items
4. A mapping from variables to items
First-Order Domains

1. A typed domain of items:

Definition (Typed Domain)
A non-empty set $\mathcal{D}$ of items is a domain.
A typing of $\mathcal{D}$ wrt. signature $\Sigma$ is a mapping $\delta : \mathcal{D} \rightarrow T_{\Sigma}$

We require from $\mathcal{D}$ and $\delta$ that no type is empty:
for each $\tau \in T_{\Sigma}$, there is a $d \in \mathcal{D}$ with $\delta(d) = \tau$

- If $\delta(d) = \tau$, we say $d$ has type $\tau$.
- $\mathcal{D}^\tau = \{d \in \mathcal{D} \mid \delta(d) = \tau\}$ is called subdomain of type $\tau$.
- It follows that $\mathcal{D}^\tau \neq \emptyset$ for each $\tau \in T_{\Sigma}$. 
First-Order States

2. A mapping from function symbol to functions on items
3. A mapping from predicate symbol to relation on items

Definition (Interpretation, First-Order State)

Let $D$ be a domain with typing $\delta$.
Let $I$ be a mapping, called interpretation, from function and predicate symbols to functions and relations on items, respectively, such that

$I(f) : D^{\tau_1} \times \cdots \times D^{\tau_r} \rightarrow D^\tau$ when $\alpha_\Sigma(f) = (\tau_1, \ldots, \tau_r, \tau)$
$I(p) \subseteq D^{\tau_1} \times \cdots \times D^{\tau_r}$ when $\alpha_\Sigma(p) = (\tau_1, \ldots, \tau_r)$

Then $S = (D, \delta, I)$ is a first-order state.
Example

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

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

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

One of uncountably many possible first-order states!
Definition

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

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: \( D = \{ o \} \)

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

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, $d \in D^\tau$:

$$\beta_y^d(x) := \begin{cases} 
\beta(x) & x \neq y \\
d & x = y 
\end{cases}$$
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)**

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

- $val_{S, \beta}(x) = \beta(x)$
- $val_{S, \beta}(f(t_1, \ldots, t_r)) = I(f)(val_{S, \beta}(t_1), \ldots, val_{S, \beta}(t_r))$
Example

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

\[
\begin{align*}
\mathcal{I}(i) &= 17 \\
\mathcal{I}(j) &= 17 \\
\mathcal{I}(f) &= \begin{array}{c|c}
\mathcal{D} & \mathcal{I}(f) \\
\hline
\text{int} & 2 \\
\text{short} & 17 \\
\text{int} & 17 \\
\end{array}
\end{align*}
\]

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

\( val_{S,\beta}(\phi) \) for \( \phi \in \text{For} \)

\( val_{S,\beta}(p(t_1, \ldots, t_r)) = T \quad \text{iff} \quad (val_{S,\beta}(t_1), \ldots, val_{S,\beta}(t_r)) \in \mathcal{I}(p) \)

\( val_{S,\beta}(\phi \land \psi) = T \quad \text{iff} \quad val_{S,\beta}(\phi) = T \text{ and } val_{S,\beta}(\psi) = T \)

\( \neg, \lor, \rightarrow, \leftrightarrow \) as in propositional logic

\( val_{S,\beta}(\forall \tau \ x; \phi) = T \quad \text{iff} \quad val_{S,\beta_{\neg x}}(\phi) = T \text{ for all } d \in \mathcal{D}^\tau \)

\( val_{S,\beta}(\exists \tau \ x; \phi) = T \quad \text{iff} \quad val_{S,\beta_{\neg x}}(\phi) = T \text{ for at least one } d \in \mathcal{D}^\tau \)
Example

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

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

<table>
<thead>
<tr>
<th>( \mathcal{D}_{\text{int}} )</th>
<th>( \mathcal{I}(f) )</th>
<th>in ( \mathcal{I}(\langle \rangle) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>F</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>(2,2)</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>(2,17)</td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>(17,2)</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>(17,17)</td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>

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

\[ \text{val}_{S,\beta}(\phi) = T \]

- \( S \models \phi \) iff for all \( \beta : \text{val}_{S,\beta}(\phi) = T \) (\( \phi \) is satisfiable)
- \( \models \phi \) iff for all \( S : S \models \phi \) (\( \phi \) is true in \( S \))
- \( \models \phi \) iff for all \( S : S \models \phi \) (\( \phi \) is valid)

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