Software Engineering using Formal Methods

First-Order Logic

Wolfgang Ahrendt & Josef Svenningsson & Meng Wang

27st September 2012

Install the KeY-Tool...

KeY used in Friday's exercise

Requires: Java ≥ 5

Follow instructions on course page, under:
⇒Links, Papers, and Software
⇒Go to KeY-SEFM2011 Version

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 application `javaws`

Motivation for Introducing First-Order Logic

we will specify JAVA programs with Java Modeling Language (JML)

JML combines

- JAVA expressions
- First-Order Logic (FOL)

we will verify JAVA programs using Dynamic Logic

Dynamic Logic combines

- First-Order Logic (FOL)
- JAVA programs

FOL: Language and Calculus

we introduce:

- FOL as a language
- (no formal semantics)
- calculus for proving FOL formulas
- KeY system as FOL prover (to start with)

First-Order Logic: Signature

Signature

A first-order signature Σ consists of

- a set T_Σ of types
- a set F_Σ of function symbols
- a set P_Σ of predicate symbols
- a typing α_Σ

intuitively, the typing α_Σ 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

$T_{\Sigma_1} = \{\mathtt{int}\}$,
$F_{\Sigma_1} = \{\mathtt{+}, \mathtt{-}\} \cup \{\ldots, \mathtt{-2}, \mathtt{-1}, \mathtt{0}, \mathtt{1}, \mathtt{2}, \ldots\}$,
$P_{\Sigma_1} = \{\mathtt{<}\}$

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

Constants

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

here, the constants are: $\ldots, \mathtt{-2}, \mathtt{-1}, \mathtt{0}, \mathtt{1}, \mathtt{2}, \ldots$

Syntax of First-Order Logic: Signature Cont'd

Type declaration of signature symbols

- Write $\tau\ x$; to declare variable x of type τ
- 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;`
Predicates	`isEmpty(List); alertOn;`
Functions	`int arrayLookup(int); Object o;`

Example Signature 1 + Notation

typing of Signature 1:

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

can alternatively be written as:

```
<(int,int);
int +(int,int);
int 0;  int 1;  int -1;  ...
```

Example Signature 2

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

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 $\alpha_\Sigma(v) \in T_\Sigma$.

Terms are defined recursively:

Terms

A first-order term of type $\tau \in T_\Sigma$

- is either a variable of type τ, or
- has the form $f(t_1, \ldots, t_n)$,
  where $f \in F_\Sigma$ has result type τ, and each t_i is term of the correct type, following the typing α_Σ of f.

If f is a constant symbol, the term is written f, instead of $f()$.

Terms over Signature 1

example terms over Σ_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 Σ_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`

Atomic Formulas

Logical Atoms

Given a signature Σ.
A logical atom has either of the forms

- *true*
- *false*
- $t_1 = t_2$ *("equality")*,
  where t_1 and t_2 have 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 α_Σ of p.

Atomic Formulas over Signature 1

example formulas over Σ_1:
(assume variable `int` v;)

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

Atomic Formulas over Signature 2

example formulas over Σ_2:
(assume variables `LinkedIntList` o; `int` v;)

- $\texttt{new}(13, \texttt{null}) = \texttt{null}$
- $\texttt{elem}(\texttt{new}(13, \texttt{null})) = 13$
- $\texttt{next}(\texttt{new}(13, \texttt{null})) = \texttt{null}$
- $\texttt{next}(\texttt{next}(o)) = o$

General Formulas

first-order formulas are defined recursively:

Formulas

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

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'.

General 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	$\forall x; \forall y; \forall z;$
	$(x < y \wedge y < z \rightarrow x < z)$

(is any of the three formulas redundant?)

Semantics (briefly, but see Appendix)

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 ϕ and ψ be arbitrary, closed formulas (whether valid of not).

The following formulas are valid:

- $\neg(\phi \wedge \psi) \leftrightarrow \neg\phi \vee \neg\psi$
- $\neg(\phi \vee \psi) \leftrightarrow \neg\phi \wedge \neg\psi$
- $(\mathit{true} \wedge \phi) \leftrightarrow \phi$
- $(\mathit{false} \vee \phi) \leftrightarrow \phi$
- $\mathit{true} \vee \phi$
- $\neg(\mathit{false} \wedge \phi)$
- $(\phi \rightarrow \psi) \leftrightarrow (\neg\phi \vee \psi)$
- $\phi \rightarrow \mathit{true}$
- $\mathit{false} \rightarrow \phi$
- $(\mathit{true} \rightarrow \phi) \leftrightarrow \phi$
- $(\phi \rightarrow \mathit{false}) \leftrightarrow \neg\phi$

Useful Valid Formulas

Assume that x is the only variable which may appear freely in ϕ or ψ.

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

	Text book	SPIN	KeY
Negation	$\neg$	`!`	`!`
Conjunction	$\wedge$	`&&`	`&`
Disjunction	$\vee$	`||`	`|`
Implication	$\rightarrow$, $\supset$	`->`	`->`
Equivalence	$\leftrightarrow$	`<->`	`<->`
Universal Quantifier	$\forall\, x; \phi$	n/a	`\forall` $\tau\ x; \phi$
Existential Quantifier	$\exists\, x; \phi$	n/a	`\exists` $\tau\ x; \phi$
Value equality	$\doteq$	`==`	`=`

Part I

Sequent Calculus for FOL

Reasoning by Syntactic Transformation

Prove Validity of ϕ by syntactic transformation of ϕ

Logic Calculus: Sequent Calculus based on notion of sequent:

$$\underbrace{\psi_1, \ldots, \psi_m}_{\text{Antecedent}} \quad \Longrightarrow \quad \underbrace{\phi_1, \ldots, \phi_n}_{\text{Succedent}}$$

has same meaning as

$$(\psi_1 \land \cdots \land \psi_m) \quad \rightarrow \quad (\phi_1 \lor \cdots \lor \phi_n)$$

which has same meaning (for closed formulas ψ_i, ϕ_i) as

$$\{\psi_1, \ldots, \psi_m\} \quad \models \quad \phi_1 \lor \cdots \lor \phi_n$$

Notation for Sequents

$$\psi_1, \ldots, \psi_m \quad \Longrightarrow \quad \phi_1, \ldots, \phi_n$$

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 a single schematic sequent

$$\Gamma \quad \Longrightarrow \quad \phi \wedge \psi, \Delta$$

Matches any sequent with occurrence of conjunction in succedent

Call $\phi \wedge \psi$ main formula and Γ, Δ 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

$$\text{RuleName} \; \frac{\overbrace{\Gamma_1 \Longrightarrow \Delta_1 \quad \cdots \quad \Gamma_r \Longrightarrow \Delta_r}^{\text{Premisses}}}{\underbrace{\Gamma \Longrightarrow \Delta}_{\text{Conclusion}}}$$

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

Example

$$\text{andRight} \; \frac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \wedge \psi, \Delta}$$

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

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

‘Propositional’ Sequent Calculus Rules

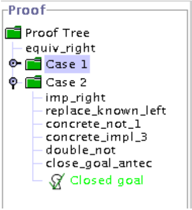
main	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \Longrightarrow \phi, \Delta}{\Gamma, \neg\phi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \phi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\phi, \Delta}$
and	$\dfrac{\Gamma, \phi, \psi \Longrightarrow \Delta}{\Gamma, \phi \wedge \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \wedge \psi, \Delta}$
or	$\dfrac{\Gamma, \phi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \vee \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \phi, \psi, \Delta}{\Gamma \Longrightarrow \phi \vee \psi, \Delta}$
imp	$\dfrac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \to \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \phi \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \to \psi, \Delta}$

close $\dfrac{}{\Gamma, \phi \Longrightarrow \phi, \Delta}$ true $\dfrac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$ false $\dfrac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$

Sequent Calculus Proofs

Goal to prove: $\mathcal{G} = \quad \psi_1, \ldots, \psi_m \implies \phi_1, \ldots, \phi_n$

- find rule $\mathcal{R}$ whose conclusion matches $\mathcal{G}$
- instantiate $\mathcal{R}$ such that conclusion identical to $\mathcal{G}$
- recursively find proofs for resulting premisses $\mathcal{G}_1, \ldots, \mathcal{G}_r$
- 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

$$
\dfrac{\dfrac{\dfrac{\textsc{Close}\;\dfrac{*}{p \Longrightarrow q,\, p} \qquad \dfrac{*}{p,\, q \Longrightarrow q}\;\textsc{Close}}{p,\, (p \rightarrow q) \Longrightarrow q}}{p \land (p \rightarrow q) \Longrightarrow q}}{\Longrightarrow (p \land (p \rightarrow q)) \rightarrow q}
$$

A proof is closed iff all its branches are closed

Demo

`prop.key`

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	$\forall\, \mathbf{int}\, x;\ (\mathrm{even}(x) \rightarrow \mathrm{divByTwo}(x))$
Let c be an arbitrary number	Declare "unused" constant $\mathbf{int}\ \mathtt{c}$
The even number c is divisible by 2	prove $\mathrm{even}(c) \rightarrow \mathrm{divByTwo}(c)$

Sequent rule $\forall$-right

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

- $[x/c]\,\phi$ is result of replacing each occurrence of x in ϕ with c
- c **new** constant of type τ

Proving Validity of First-Order Formulas Cont'd

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	$\exists\, \mathbf{int}\, x;$ prime(x)
Provide any "witness", say, 7	Use variable-free term $\mathbf{int}\ 7$
7 is a prime number	prime(7)

Sequent rule $\exists$-right

$$\text{existsRight}\ \frac{\Gamma \Longrightarrow [x/t]\,\phi,\ \exists \tau\, x;\ \phi, \Delta}{\Gamma \Longrightarrow \exists \tau\, x;\ \phi, \Delta}$$

- t any variable-free term of type τ
- 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	$\forall\, \mathbf{int}\, x;\ (\mathrm{prime}(x) \rightarrow \mathrm{odd}(x))$
In particular, this holds for 17	Use variable-free term $\mathbf{int}$ 17
We know: if 17 is prime it is odd	$\mathrm{prime}(17) \rightarrow \mathrm{odd}(17)$

Sequent rule $\forall$-left

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

- t' any variable-free term of type τ
- We might need other instances besides t'! Keep premise $\forall \tau\, x;\ \phi$

Proving Validity of First-Order Formulas Cont'd

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

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

- c **new** constant of type τ

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

Let x, y denote integer constants, both are not zero. We know further that x divides y.
Show: $(y/x) * x \doteq y$ ($'/'$ 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 $k * x \doteq y$.
Let now c denote such a k. Hence we can replace y by $c * x$ on the right side (see slide 35). ... □

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdots}}{\neg(x \doteq 0), \neg(y \doteq 0), c * x \doteq y \Longrightarrow ((c * x)/x) * x \doteq y}}{\neg(x \doteq 0), \neg(y \doteq 0), c * x \doteq y \Longrightarrow (y/x) * x \doteq y}}{\neg(x \doteq 0), \neg(y \doteq 0), \exists\, \texttt{int}\ k; k * x \doteq y \Longrightarrow (y/x) * x \doteq y}$$

Proving Validity of First-Order Formulas Cont'd

Example (A simple theorem about binary relations)

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{*}{p(c,d),\ \forall\, y;\ p(c,y) \Longrightarrow p(c,d),\ \exists\, x;\ p(x,y)}}{p(c,d),\ \forall\, y;\ p(c,y) \Longrightarrow \exists\, x;\ p(x,d)}}{\forall\, y;\ p(c,y) \Longrightarrow \exists\, x;\ p(x,d)}}{\forall\, y;\ p(c,y) \Longrightarrow \forall\, y;\ \exists\, x;\ p(x,y)}}{\exists\, x;\ \forall\, y;\ p(x,y) \Longrightarrow \forall\, y;\ \exists\, x;\ p(x,y)}$$

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

Demo

Proving Validity of First-Order Formulas Cont'd

Using an equation between terms

We assume $t \doteq t'$ is true

How is such a fact used in a mathematical proof?

Use $x \doteq y-1$ to simplify $x+1/y$ $\qquad x \doteq y-1 \Longrightarrow 1 \doteq x+1/y$

Replace x in conclusion with right-hand side of equation

We know: $x+1/y$ equal to $y-1+1/y$ $\qquad x \doteq y-1 \Longrightarrow 1 \doteq y-1+1/y$

Sequent rule $\doteq$-left

$$\text{applyEqL}\ \frac{\Gamma, t \doteq t', [t/t']\,\phi \Longrightarrow \Delta}{\Gamma, t \doteq t', \phi \Longrightarrow \Delta} \qquad \text{applyEqR}\ \frac{\Gamma, t \doteq t' \Longrightarrow [t/t']\,\phi, \Delta}{\Gamma, t \doteq t' \Longrightarrow \phi, \Delta}$$

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

Proving Validity of First-Order Formulas Cont'd

Closing a subgoal in a proof

- We derived a sequent that is obviously valid

$$\text{close}\ \frac{}{\Gamma, \phi \Longrightarrow \phi, \Delta} \qquad \text{true}\ \frac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta} \qquad \text{false}\ \frac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$$

- We derived an equation that is obviously valid

$$\text{eqClose}\ \frac{}{\Gamma \Longrightarrow t \doteq t, \Delta}$$

Sequent Calculus for FOL at One Glance

	left side, antecedent	right side, succedent
$\forall$	$\dfrac{\Gamma, \forall \tau\, x;\ \phi,\ [x/t']\,\phi \Longrightarrow \Delta}{\Gamma, \forall \tau\, x;\ \phi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow [x/c]\,\phi, \Delta}{\Gamma \Longrightarrow \forall \tau\, x;\ \phi, \Delta}$
$\exists$	$\dfrac{\Gamma, [x/c]\,\phi \Longrightarrow \Delta}{\Gamma, \exists \tau\, x;\ \phi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow [x/t']\,\phi,\ \exists \tau\, x;\ \phi, \Delta}{\Gamma \Longrightarrow \exists \tau\, x;\ \phi, \Delta}$
$\doteq$	$\dfrac{\Gamma, t \doteq t' \Longrightarrow [t/t']\,\phi, \Delta}{\Gamma, t \doteq t' \Longrightarrow \phi, \Delta}$ (+ application rule on left side)	$\dfrac{}{\Gamma \Longrightarrow t \doteq t, \Delta}$

- $[t/t']\,\phi$ is result of replacing each occurrence of t in ϕ with t'
- t, t' variable-free terms of type τ
- c **new** constant of type τ (occurs not on current proof branch)
- Equations can be reversed by commutativity

Recap: 'Propositional' Sequent Calculus Rules

main	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \Longrightarrow \phi, \Delta}{\Gamma, \neg\phi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \phi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\phi, \Delta}$
and	$\dfrac{\Gamma, \phi, \psi \Longrightarrow \Delta}{\Gamma, \phi \wedge \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \wedge \psi, \Delta}$
or	$\dfrac{\Gamma, \phi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \vee \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \phi, \psi, \Delta}{\Gamma \Longrightarrow \phi \vee \psi, \Delta}$
imp	$\dfrac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \rightarrow \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \phi \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \rightarrow \psi, \Delta}$

close $\dfrac{}{\Gamma, \phi \Longrightarrow \phi, \Delta}$ true $\dfrac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$ false $\dfrac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$

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

essential:

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

further reading:

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

KeYbook B. Beckert, R. Hähnle, and P. Schmitt, editors, Verification of Object-Oriented Software: The KeY Approach, vol 4334 of *LNCS* (Lecture Notes in Computer Science), Springer, 2006 (access via Chalmers library → E-books → Lecture Notes in Computer Science)

Part II

Appendix: 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 object
- Respect typing: `int i`, `List l` must denote different objects

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

1. A collection of typed universes of objects
2. A mapping from variables to objects
3. A mapping from function arguments to function values
4. The set of argument tuples where a predicate is true

First-Order Domains/Universes

1. A collection of typed universes of objects

Definition (Universe/Domain)

A non-empty set $\mathcal{D}$ of objects is a universe or domain
Each element of $\mathcal{D}$ has a fixed type given by $\delta : \mathcal{D} \to \tau$

- 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 δ

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 $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ 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

$$\mathcal{I}(i) = 17$$
$$\mathcal{I}(j) = 17$$
$$\mathcal{I}(\texttt{obj}) = o$$

$\mathcal{D}^{\text{int}}$	$\mathcal{I}(f)$
2	2
17	2

$\mathcal{D}^{\text{int}} \times \mathcal{D}^{\text{int}}$	in $\mathcal{I}(<)$?
$(2, 2)$	F
$(2, 17)$	T
$(17, 2)$	F
$(17, 17)$	F

One of uncountably many possible first-order states!

Semantics of Reserved Signature Symbols

Definition

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

Interpretation is fixed as $\mathcal{I}(\doteq) = \{(d, d) \mid d \in \mathcal{D}\}$
"Referential Equality" (holds if arguments refer to identical object)

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 $\mathcal{I}(\texttt{obj1}) = \mathcal{I}(\texttt{obj2}) = o$

Variable Assignments

2. A mapping from variables to objects

Think of variable assignment as environment for storage of local variables

Definition (Variable Assignment)

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

Definition (Modified Variable Assignment)

Let y be variable of type τ, β variable assignment, $d \in \mathcal{D}^\tau$:

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

Semantic Evaluation of Terms

Given a first-order state $\mathcal{S}$ and a variable assignment β
it is possible to evaluate first-order terms under $\mathcal{S}$ and β

Definition (Valuation of Terms)

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

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

Semantic Evaluation of Terms Cont'd

Example

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

$\mathcal{I}(\texttt{i}) = 17$
$\mathcal{I}(\texttt{j}) = 17$

$\mathcal{D}^{\textbf{int}}$	$\mathcal{I}(\texttt{f})$
2	17
17	2

Var	β
obj	o
x	17

- $val_{\mathcal{S},\beta}(\texttt{f(f(i))})$?
- $val_{\mathcal{S},\beta}(x)$?

Semantic Evaluation of Formulas

Definition (Valuation of Formulas)

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

- $val_{\mathcal{S},\beta}(p(t_1, \ldots, t_r) = T$ iff $(val_{\mathcal{S},\beta}(t_1), \ldots, val_{\mathcal{S},\beta}(t_r)) \in \mathcal{I}(p)$
- $val_{\mathcal{S},\beta}(\phi \wedge \psi) = T$ iff $val_{\mathcal{S},\beta}(\phi) = T$ and $val_{\mathcal{S},\beta}(\psi) = T$
- ... as in propositional logic
- $val_{\mathcal{S},\beta}(\forall\, \tau\ x;\ \phi) = T$ iff $val_{\mathcal{S},\beta_x^d}(\forall\, \tau\ x;\ \phi) = T$ for all $d \in \mathcal{D}^\tau$
- $val_{\mathcal{S},\beta}(\forall\, \tau\ x;\ \phi) = T$ iff $val_{\mathcal{S},\beta_x^d}(\forall\, \tau\ x;\ \phi) = T$ for at least one $d \in \mathcal{D}^\tau$

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

$\mathcal{I}(j) = 17$

$\mathcal{I}(\texttt{obj}) = o$

$\mathcal{D}^{\mathbf{int}}$	$\mathcal{I}(f)$
2	2
17	2

$\mathcal{D}^{\mathbf{int}} \times \mathcal{D}^{\mathbf{int}}$	in $\mathcal{I}(<)$?
$(2, 2)$	F
$(2, 17)$	T
$(17, 2)$	F
$(17, 17)$	F

- $val_{\mathcal{S},\beta}(f(j) < j)$?
- $val_{\mathcal{S},\beta}(\exists\, \mathbf{int}\ x;\ f(x) \doteq x)$?
- $val_{\mathcal{S},\beta}(\forall\, \texttt{Object}\ o1;\ \forall\, \texttt{Object}\ o2;\ o1 \doteq o2)$?

Semantic Notions

Definition (Satisfiability, Truth, Validity)

$val_{\mathcal{S},\beta}(\phi) = T$			(ϕ is **satisfiable**)
$\mathcal{S} \models \phi$	iff	for all β : $val_{\mathcal{S},\beta}(\phi) = T$	(ϕ is **true** in $\mathcal{S}$)
$\models \phi$	iff	for all $\mathcal{S}$: $\mathcal{S} \models \phi$	(ϕ is **valid**)

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

Example

- $f(j) < j$ is true in $\mathcal{S}$
- $\exists\, \mathbf{int}\ x;\ i \doteq x$ is valid
- $\exists\, \mathbf{int}\ x;\ \neg(x \doteq x)$ is not satisfiable