UNSW SYDNEY

COMP4161

Advanced Topics in Software Verification

HOL

Gerwin Klein, June Andronick, Miki Tanaka, Johannes Åman Pohjola

T3/2022

Content

➜ Foundations & Principles

- Intro, Lambda [illegible] atural deduction [1,2]
- Higher Order [illegible] part 1) [2,3[a]]
- Term rewriting [3,4]

➜ Proof & Specification Techniques

- Inductively defined sets, rule induction [4,5]
- Datatype induction, primitive recursion [5,7]
- General recursive functions, termination proofs [7[b]]
- Proof automation, Isar (part 2) [8]
- Hoare logic, proofs about programs, invariants [8,9]
- C verification [9,10]
- Practice, questions, exam prep [10[c]]

[a] a1 due; [b] a2 due; [c] a3 due

More on Automation

Last time: safe and unsafe. heuristics: use safe before unsafe

More on Automation

Last time: safe and unsafe. heuristics: use safe before unsafe

be automated

More on Automation

Last time: safe and unsafe. heuristics: use safe before unsafe

be automated

Automated metho last, clarify etc) are not hardwired.

Safe/uns lim rules can be declared.

More on Automation

Last time: safe and unsafe, heuristics: use safe before unsafe

be automated

Automated metho last, clarify etc) are not hardwired.

Safe/uns lim rules can be declared.

Syntax:

[<kind>!] for safe rules (<kind> one of intro, elim, dest)

[<kind>] for unsafe rules

More on Automation

Last time: safe and unsafe, heuristics: use safe before unsafe

be automated

Automated metho last, clarify etc) are not hardwired.

Safe/uns lim rules can be declared.

Syntax:

[<kind>!] for safe rules (<kind> one of intro, elim, dest)

[<kind>] for unsafe rules

Application (roughly):

do safe rules first, search/backtrack on unsafe rules only

More on Automation

Last time: safe and unsafe, heuristics: use safe before unsafe

be automated

Automated metho last, clarify etc) are not hardwired.

Safe/uns lim rules can be declared.

Syntax:

[<kind>!] for safe rules (<kind> one of intro, elim, dest)

[<kind>] for unsafe rules

Application (roughly):

do safe rules first, search/backtrack on unsafe rules only

Example:

declare attribute globally	**declare** conjI [intro!] allE [elim]
remove attribute globally	**declare** allE [rule del]
use locally	**apply** (blast intro: someI)
delete locally	**apply** (blast del: conjI)

Demo: Automation

Exercises

- derive the classical contradiction rule $(\neg P \Longrightarrow \mathit{False}) \Longrightarrow P$ in Isabelle
- define **nor** and **nand**
- show nor x x $=$
- derive safe intro and for them
- use these in an automated proof of nor x x $=$ nand x x

Defining Higher Order Logic

What is Higher Order Logic?

- **Propositional Logi**
  - no quantifiers
  - all variables ha bl

What is Higher Order Logic?

- **Propositional Logi**
  - no quantifiers
  - all variables ha bl
- **First Order Logic:**
  - quantification over values, but not over functions and predicates,
  - terms and formulas syntactically distinct

What is Higher Order Logic?

- **Propositional Logi**
  - no quantifiers
  - all variables ha ol
- **First Order Logic:**
  - quantification over values, but not over functions and predicates,
  - terms and formulas syntactically distinct
- **Higher Order Logic:**
  - quantification over everything, including predicates
  - consistency by types
  - formula = term of type bool
  - definition built on $\lambda^{\rightarrow}$ with certain default types and constants

Defining Higher Order Logic

Default types:

Defining Higher Order Logic

Default types:

bool

Defining Higher Order Logic

Default types:

bool  $\Rightarrow$ _

Defining Higher Order Logic

Default types:

bool _ ⇒ _ ind

Defining Higher Order Logic

Default types:

bool _ ⇒ _ ind

- **bool** sometimes called *o*
- ⇒ sometimes called *fun*

Defining Higher Order Logic

Default types:

bool _ ⇒ _ ind

- **bool** sometimes called *o*
- ⇒ sometimes called *fun*

Default Constants:

Defining Higher Order Logic

Default types:

bool _ ⇒ _ ind

- **bool** sometimes called *o*
- ⇒ sometimes called *fun*

Default Constants:

$\longrightarrow$ ⊥ : *bool* ⇒ *bool* ⇒ *bool*

Defining Higher Order Logic

Default types:

bool _ ⇒ _ ind

- **bool** sometimes called *o*
- ⇒ sometimes called *fun*

Default Constants:

$$\longrightarrow :: bool \Rightarrow bool \Rightarrow bool$$

$$= :: \alpha \Rightarrow \alpha \Rightarrow bool$$

Defining Higher Order Logic

Default types:

bool _ ⇒ _ ind

- **bool** sometimes called *o*
- ⇒ sometimes called *fun*

Default Constants:

$$\longrightarrow :: bool \Rightarrow bool \Rightarrow bool$$

$$= :: \alpha \Rightarrow \alpha \Rightarrow bool$$

$$\epsilon :: (\alpha \Rightarrow bool) \Rightarrow \alpha$$

Higher Order Abstract Syntax

Problem: Define syntax for binders like $\forall$, $\exists$, ε

Higher Order Abstract Syntax

Problem: Define syntax for binders like $\forall$, $\exists$, ε

One approach: $\forall :: \mathit{va} \Rightarrow \mathit{bool}$

Drawback: need to th substitution, α conversion again.

Higher Order Abstract Syntax

Problem: Define syntax for binders like $\forall$, $\exists$, ε

One approach: $\forall :: \mathit{va} \Rightarrow \mathit{bool}$
Drawback: need to th substitution, α conversion again.

But: Already have binder, substitution, α conversion in meta logic

Higher Order Abstract Syntax

Problem: Define syntax for binders like $\forall$, $\exists$, ε

One approach: $\forall :: \mathit{va} \Rightarrow \mathit{bool}$
Drawback: need to th substitution, α conversion again.

But: Already have binder, substitution, α conversion in meta logic

So: Use λ to encode all other binders.

Higher Order Abstract Syntax

Example:

$\Rightarrow bool) \Rightarrow bool$

HOAS **usual syntax**

Higher Order Abstract Syntax

Example:

$\Rightarrow bool) \Rightarrow bool$

HOAS **usual syntax**

ALL $(\lambda x.\ x = 2)$

Higher Order Abstract Syntax

Example:

$\Rightarrow bool) \Rightarrow bool$

HOAS	**usual syntax**
ALL $(\lambda x.\ x = 2)$	$\forall x.\ x = 2$

Higher Order Abstract Syntax

Example:

$\Rightarrow bool) \Rightarrow bool$

HOAS	**usual syntax**
ALL $(\lambda x.\ x = 2)$	$\forall x.\ x = 2$
ALL P	

Higher Order Abstract Syntax

Example:

$\Rightarrow bool) \Rightarrow bool$

HOAS	**usual syntax**
ALL $(\lambda x.\ x = 2)$	$\forall x.\ x = 2$
ALL P	$\forall x.\ P\ x$

Higher Order Abstract Syntax

Example:

$\Rightarrow bool) \Rightarrow bool$

HOAS	**usual syntax**
ALL $(\lambda x.\ x = 2)$	$\forall x.\ x = 2$
ALL P	$\forall x.\ P\ x$

Isabelle can translate usual binder syntax into HOAS.

Side Track: Syntax Declarations

➜ **mixfix:**
consts drvbl :: *ct*  ⇒ *bool* ("_, _ ⊢ _")
Legal syntax now:

Side Track: Syntax Declarations

- **mixfix:**
  consts drvbl :: *ct* ⇒ *bool* ("_, _ ⊢ _")
  Legal syntax now:
- **priorities:**
  pattern can be annotated with priorities to indicate binding strength
  Example: drvbl :: *ct* ⇒ *ct* ⇒ *fm* ⇒ *bool* ("_, _ ⊢ _" [30, 0, 20] 60)

Side Track: Syntax Declarations

- **mixfix:**
  consts drvbl :: *ct* ⇒ *bool* (”_, _ ⊢ _”)
  Legal syntax now:
- **priorities:**
  pattern can be annotated with priorities to indicate binding strength
  Example: drvbl :: *ct* ⇒ *ct* ⇒ *fm* ⇒ *bool* (”_, _ ⊢ _” [30, 0, 20] 60)
- **infixl/infixr**: short form for left/right associative binary operators
  Example: or :: *bool* ⇒ *bool* ⇒ *bool* (infixr ”∨” 30)

Side Track: Syntax Declarations

- **mixfix:**
  consts drvbl :: *ct* ⇒ *ct* ⇒ *fm* ⇒ *bool* ("_, _ ⊢ _")
  Legal syntax now:
- **priorities:**
  pattern can be annotated with priorities to indicate binding strength
  Example: drvbl :: *ct* ⇒ *ct* ⇒ *fm* ⇒ *bool* ("_, _ ⊢ _" [30, 0, 20] 60)
- **infixl/infixr**: short form for left/right associative binary operators
  Example: or :: *bool* ⇒ *bool* ⇒ *bool* (infixr "∨" 30)
- **binders:** declaration must be of the form
  $c :: (\tau_1 \Rightarrow \tau_2) \Rightarrow \tau_3$ (binder "B" < p >)
  B x. *P x* translated into *c P* (and vice versa)
  Example ALL :: $(\alpha \Rightarrow bool) \Rightarrow bool$ (binder "∀" 10)

Side Track: Syntax Declarations

- **mixfix:**
  consts drvbl :: *ct* ⇒ *ct* ⇒ *fm* ⇒ *bool* ("_, _ ⊢ _")
  Legal syntax now:
- **priorities:**
  pattern can be annotated with priorities to indicate binding strength
  Example: drvbl :: *ct* ⇒ *ct* ⇒ *fm* ⇒ *bool* ("_, _ ⊢ _" [30, 0, 20] 60)
- **infixl/infixr**: short form for left/right associative binary operators
  Example: or :: *bool* ⇒ *bool* ⇒ *bool* (infixr "∨" 30)
- **binders:** declaration must be of the form
  $c :: (\tau_1 \Rightarrow \tau_2) \Rightarrow \tau_3$ (binder "B" $< p >$)
  $B\ x.\ P\ x$ translated into $c\ P$ (and vice versa)
  Example ALL :: $(\alpha \Rightarrow bool) \Rightarrow bool$ (binder "∀" 10)

More in Isabelle/Isar Reference Manual (8.2)

Back to HOL

Base: $bool$, $\Rightarrow$, ind $=$, $\longrightarrow$, ε

And the rest is

Back to HOL

Base: $bool$, $\Rightarrow$, ind, $=$, $\longrightarrow$, ε

And the rest is defin

$\mathsf{True} \equiv$

$\mathsf{All}\ P \equiv$

$\mathsf{Ex}\ P \equiv$

$\mathsf{False} \equiv$

$\neg P \equiv$

$P \wedge Q \equiv$

$P \vee Q \equiv$

$\mathsf{If}\ P\ x\ y \equiv$

$\mathsf{inj}\ f \equiv$

$\mathsf{surj}\ f \equiv$

Back to HOL

Base: $bool$, $\Rightarrow$, ind, $=$, $\longrightarrow$, ε

And the rest is defin

True	$\equiv$	$(\lambda x :: \ldots. (\lambda x.\ x)$
All P	$\equiv$	
Ex P	$\equiv$	
False	$\equiv$	
$\neg P$	$\equiv$	
$P \wedge Q$	$\equiv$	
$P \vee Q$	$\equiv$	
If $P\ x\ y$	$\equiv$	
inj f	$\equiv$	
surj f	$\equiv$	

Back to HOL

Base: $bool, \Rightarrow, ind, =, \longrightarrow, \varepsilon$

And the rest is defin

True	$\equiv$	$(\lambda x :: \ldots. x) = (\lambda x. x)$
All P	$\equiv$	$P = (\lambda x. \text{True})$
Ex P	$\equiv$	
False	$\equiv$	
$\neg P$	$\equiv$	
$P \wedge Q$	$\equiv$	
$P \vee Q$	$\equiv$	
If $P\ x\ y$	$\equiv$	
inj f	$\equiv$	
surj f	$\equiv$	

Back to HOL

Base: $bool,\ \Rightarrow,\ ind \qquad =,\ \longrightarrow,\ \varepsilon$

And the rest is defin

$$
\begin{array}{lcl}
\mathsf{True} & \equiv & (\lambda x :: bool.\ x) = (\lambda x.\ x) \\
\mathsf{All}\ P & \equiv & P = (\lambda x.\ \mathsf{True}) \\
\mathsf{Ex}\ P & \equiv & \forall Q.\ (\forall x.\ P\ x \longrightarrow Q) \longrightarrow Q \\
\mathsf{False} & \equiv & \forall P.\ P \\
\neg P & \equiv & P \longrightarrow \mathsf{False} \\
P \wedge Q & \equiv & \forall R.\ (P \longrightarrow Q \longrightarrow R) \longrightarrow R \\
P \vee Q & \equiv & \forall R.\ (P \longrightarrow R) \longrightarrow (Q \longrightarrow R) \longrightarrow R \\
\mathsf{If}\ P\ x\ y & \equiv & \mathsf{SOME}\ z.\ (P = \mathsf{True} \longrightarrow z = x) \wedge (P = \mathsf{False} \longrightarrow z = y) \\
\mathsf{inj}\ f & \equiv & \forall x\ y.\ f\ x = f\ y \longrightarrow x = y \\
\mathsf{surj}\ f & \equiv & \forall y.\ \exists x.\ y = f\ x
\end{array}
$$

The Axioms of HOL

$$\frac{}{t = t}\ \text{refl} \qquad \frac{s = t}{t = s}\ \text{subst} \qquad \frac{\bigwedge x.\ f\ x = g\ x}{(\lambda x.\ f\ x) = (\lambda x.\ g\ x)}\ \text{ext}$$

The Axioms of HOL

$$\frac{}{t = t}\ \text{refl} \qquad \frac{s =}{P =}\ \text{ubst} \qquad \frac{\bigwedge x.\ f\ x = g\ x}{(\lambda x.\ f\ x) = (\lambda x.\ g\ x)}\ \text{ext}$$

$$\frac{}{P \longrightarrow Q}$$

$$\frac{P \longrightarrow Q \quad P}{Q}\ \text{mp}$$

The Axioms of HOL

$$\frac{}{t = t}\ \text{refl} \qquad \frac{s =}{}\ \text{ubst} \qquad \frac{\bigwedge x.\ f\ x = g\ x}{(\lambda x.\ f\ x) = (\lambda x.\ g\ x)}\ \text{ext}$$

$$\frac{P =}{P \longrightarrow Q} \qquad \frac{P \longrightarrow Q \quad P}{Q}\ \text{mp}$$

$$\frac{}{(P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P = Q)}\ \text{iff}$$

The Axioms of HOL

$$\frac{}{t = t}\ \text{refl} \qquad \frac{s =}{}\ \text{ubst} \qquad \frac{\bigwedge x.\ f\ x = g\ x}{(\lambda x.\ f\ x) = (\lambda x.\ g\ x)}\ \text{ext}$$

$$\frac{P =}{P \longrightarrow Q} \qquad \frac{P \longrightarrow Q \quad P}{Q}\ \text{mp}$$

$$\frac{}{(P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P = Q)}\ \text{iff}$$

$$\frac{}{P = \text{True} \vee P = \text{False}}\ \text{True\_or\_False}$$

The Axioms of HOL

$$\frac{}{t = t}\ \text{refl} \qquad \frac{s =}{}\ \text{ubst} \qquad \frac{\bigwedge x.\ f\ x = g\ x}{(\lambda x.\ f\ x) = (\lambda x.\ g\ x)}\ \text{ext}$$

$$\frac{P =}{P \longrightarrow Q} \qquad \frac{P \longrightarrow Q \quad P}{Q}\ \text{mp}$$

$$\frac{}{(P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P = Q)}\ \text{iff}$$

$$\frac{}{P = \text{True} \vee P = \text{False}}\ \text{True\_or\_False}$$

$$\frac{P\ ?x}{P\ (\text{SOME}\ x.\ P\ x)}\ \text{somel}$$

The Axioms of HOL

$$\frac{}{t = t}\ \text{refl} \qquad \frac{s = \quad}{P = \quad}\ \text{subst} \qquad \frac{\bigwedge x.\ f\ x = g\ x}{(\lambda x.\ f\ x) = (\lambda x.\ g\ x)}\ \text{ext}$$

$$\frac{P \longrightarrow Q \quad P}{Q}\ \text{mp}$$

$$\frac{}{(P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P = Q)}\ \text{iff}$$

$$\frac{}{P = \text{True} \vee P = \text{False}}\ \text{True\_or\_False}$$

$$\frac{P\ ?x}{P\ (\text{SOME}\ x.\ P\ x)}\ \text{somel}$$

$$\frac{}{\exists f :: ind \Rightarrow ind.\ \text{inj}\ f \wedge \neg \text{surj}\ f}\ \text{infty}$$

That's it.

- ➜ 3 basic constants
- ➜ 3 basic types
- ➜ 9 axioms

That's it.

- ➜ 3 basic constants
- ➜ 3 basic types
- ➜ 9 axioms

With this you can define and derive all the rest.

That's it.

- ➜ 3 basic constants
- ➜ 3 basic types
- ➜ 9 axioms

With this you can define and derive all the rest.

Isabelle knows 2 more axioms:

$$\frac{x = y}{x \equiv y}\ \text{eq\_reflection} \qquad \frac{}{(\text{THE } x.\ x = a) = a}\ \text{the\_eq\_trivial}$$

Demo: The Definitions in Isabelle

Deriving Proof Rules

In the following, we wil

Deriving Proof Rules

In the following, we wil

➜ look at the definitic detail

Deriving Proof Rules

In the following, we will

- ➜ look at the definitic detail
- ➜ derive the tradition s from the axioms in Isabelle

Deriving Proof Rules

In the following, we will

- ➜ look at the definitic detail
- ➜ derive the tradition s from the axioms in Isabelle

Convenient for deriving rules: **named assumptions in lemmas**

lemma [*name* :]
assumes [$name_1$:] "$< prop >_1$"
assumes [$name_2$:] "$< prop >_2$"
⋮
shows "$< prop >$" $< proof >$

Deriving Proof Rules

In the following, we will

- ➜ look at the definitic detail
- ➜ derive the tradition s from the axioms in Isabelle

Convenient for deriving rules: **named assumptions in lemmas**

lemma [*name* :]
assumes [$name_1$:] "$< prop >_1$"
assumes [$name_2$:] "$< prop >_2$"
⋮
shows "$< prop >$" $< proof >$

proves: $[\![< prop >_1; < prop >_2; \ldots]\!] \Longrightarrow < prop >$

True

consts True :: *bool*

True $\equiv (\lambda x :: bool.\ x)$

Intuition:

right hand side is alway

True

consts True :: *bool*
True $\equiv (\lambda x :: bool.\ x)$

Intuition:
right hand side is alway

Proof Rules:

$$\frac{}{\text{True}}\ \text{TrueI}$$

Proof:

$$\frac{\dfrac{}{(\lambda x :: bool.\ x) = (\lambda x.\ x)}\ \text{refl}}{\text{True}}\ \text{unfold True\_def}$$

Demo

Universal Quantifier

consts ALL :: $(\alpha \Rightarrow bool) \Rightarrow bool$
ALL $P \equiv P = (\lambda x.$

Intuition:

- ALL P is Higher Order Abstract Syntax for $\forall x.\ P\ x$.
- P is a function that takes an x and yields a truth value.
- ALL P should be true iff P yields true for all x, i.e. if it is equivalent to the function $\lambda x.$ True.

Proof Rules:

$$\frac{\bigwedge x.\ P\ x}{\forall x.\ P\ x}\ \text{allI} \qquad \frac{\forall x.\ P\ x \quad P\ ?x \Longrightarrow R}{R}\ \text{allE}$$

Proof: Isabelle Demo

False

consts False :: *bool*

False $\equiv \forall P.P$

Intuition:

Everything can be deri*alse*.

Proof Rules:

$$\frac{\text{False}}{P}\ \text{FalseE} \qquad \frac{}{\text{True} \neq \text{False}}$$

Proof: Isabelle Demo

Negation

consts Not :: *bool* ⇒ *bool* (¬ _)
$\neg P \equiv P \longrightarrow \text{False}$

Intuition:
Try $P = True$ and $P = False$ and the traditional truth table for $\longrightarrow$.

Proof Rules:

$$\frac{A \Longrightarrow False}{\neg A}\ \text{notI} \qquad \frac{\neg A \quad A}{P}\ \text{notE}$$

Proof: Isabelle Demo

Existential Quantifier

consts EX :: $(\alpha \Rightarrow bool) \Rightarrow bool$

EX $P \equiv \forall Q.\ (\forall x.\ P\ x \longrightarrow Q) \longrightarrow Q$

Intuition:

- ➜ EX P is HOAS for $\exists x.\ P\ x$. (like $\forall$)
- ➜ Right hand side is characterization of $\exists$ with $\forall$ and $\longrightarrow$
- ➜ Note that inner $\forall$ binds wide: $(\forall x.\ P\ x \longrightarrow Q)$
- ➜ Remember lemma from last time: $(\forall x.\ P\ x \longrightarrow Q) = ((\exists x.\ P\ x) \longrightarrow Q)$

Proof Rules:

$$\frac{P\ ?x}{\exists x.\ P\ x}\ \text{exI} \qquad \frac{\exists x.\ P\ x \quad \bigwedge x.\ P\ x \longrightarrow R}{R}\ \text{exE}$$

Proof: Isabelle Demo

Conjunction

consts And :: $bool \Rightarrow bool \Rightarrow bool$ $(\_ \wedge \_)$

$P \wedge Q \equiv \forall R.\ (P \longrightarrow (Q \longrightarrow R) \longrightarrow R$

Intuition:

- ➜ Mirrors proof rules for $\wedge$
- ➜ Try truth table for P, Q, and R

Proof Rules:

$$\frac{A \quad B}{A \wedge B}\ \text{conjI} \qquad \frac{A \wedge B \quad [\![A; B]\!] \Longrightarrow C}{C}\ \text{conjE}$$

Proof: Isabelle Demo

Disjunction

consts Or :: $bool \Rightarrow bool \Rightarrow bool$ $(\_ \vee \_)$

$P \vee Q \equiv \forall R.\ (P \longrightarrow R) \longrightarrow (Q \longrightarrow R) \longrightarrow R$

Intuition:

- ➜ Mirrors proof rules for $\vee$ (case distinction)
- ➜ Try truth table for P, Q, and R

Proof Rules:

$$\frac{A}{A \vee B}\ \frac{B}{A \vee B}\ \text{disjI1/2} \qquad \frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{C}\ \text{disjE}$$

Proof: Isabelle Demo

If-Then-Else

consts If :: $bool \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \alpha$ (if _ then _ else _)

If $P\ x\ y \equiv$ SOME $z. (P = \text{True} \longrightarrow z = x) \wedge (P = \text{False} \longrightarrow z = y)$

Intuition:

- ➜ for $P =$ True, right hand side collapses to SOME $z.\ z = x$
- ➜ for $P =$ False, right hand side collapses to SOME $z.\ z = y$

Proof Rules:

$$\frac{}{\text{if True then } s \text{ else } t = s}\ \text{ifTrue} \qquad \frac{}{\text{if False then } s \text{ else } t = t}\ \text{ifFalse}$$

Proof: Isabelle Demo

That was HOL

We have learned today ...

➜ More automation

We have learned today ...

- ➜ More automation
- ➜ Defining HOL

We have learned today ...

- ➜ More automation
- ➜ Defining HOL
- ➜ Higher Order Abstr

We have learned today ...

- ➜ More automation
- ➜ Defining HOL
- ➜ Higher Order Abstr
- ➜ Deriving proof rules