



DATA
61

COMP4161: Advanced Topics in Software Verification

HOL

Gerwin Klein, June Andronick, Christine Rizkallah, Miki Tanaka
S2/2018

data61.csiro.au



Content



- Intro & motivation, getting started [1]

- Foundations & Principles
 - Lambda Calculus, natural deduction [1,2]
 - Higher Order Logic [3^a]
 - Term rewriting [4]

- Proof & Specification Techniques
 - Inductively defined sets, rule induction [5]
 - Datatypes, recursion, induction [6, 7]
 - Hoare logic, proofs about programs, invariants [8^b,9]
 - (mid-semester break)
 - C verification [10]
 - CakeML, Isar [11^c]
 - Concurrency [12]

^aa1 due; ^ba2 due; ^ca3 due

Defining Higher Order Logic

What is Higher Order Logic?



→ Propositional Logic:

- no quantifiers
- all variables have type bool

What is Higher Order Logic?



→ Propositional Logic:

- no quantifiers
- all variables have type bool

→ First Order Logic:

- quantification over values, but not over functions and predicates,
- terms and formulas syntactically distinct

What is Higher Order Logic?



→ Propositional Logic:

- no quantifiers
- all variables have type bool

→ 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 λ^{\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

$- \Rightarrow -$

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:

`→` `::` `bool ⇒ bool ⇒ bool`

Defining Higher Order Logic



Default types:

`bool` `_ ⇒ _` `ind`

→ `bool` sometimes called `o`

→ `⇒` sometimes called `fun`

Default Constants:

`→` `::` `bool ⇒ bool ⇒ bool`
`=` `::` `α ⇒ α ⇒ bool`

Defining Higher Order Logic



Default types:

`bool` `_ ⇒ _` `ind`

- `bool` sometimes called `o`
- `⇒` sometimes called `fun`

Default Constants:

`→` :: `bool ⇒ bool ⇒ bool`
`=` :: `α ⇒ α ⇒ bool`
`ε` :: `(α ⇒ bool) ⇒ α`

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: $\forall :: \text{var} \Rightarrow \text{term} \Rightarrow \text{bool}$

Drawback: need to think about substitution, α conversion again.

Higher Order Abstract Syntax



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

One approach: $\forall :: var \Rightarrow term \Rightarrow bool$

Drawback: need to think about substitution, α conversion again.

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

λ

Higher Order Abstract Syntax



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

One approach: $\forall :: var \Rightarrow term \Rightarrow bool$

Drawback: need to think about substitution, α conversion again.

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

λ

So: Use λ to encode all other binders.

Higher Order Abstract Syntax



Example:

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

HOAS

usual syntax

Higher Order Abstract Syntax



Example:

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

HOAS

usual syntax

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

Higher Order Abstract Syntax



Example:

$$\text{ALL} :: (\alpha \Rightarrow \text{bool}) \Rightarrow \text{bool}$$

HOAS

usual syntax

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

$\forall x. x = 2$

Higher Order Abstract Syntax



Example:

$$\text{ALL} :: (\alpha \Rightarrow \text{bool}) \Rightarrow \text{bool}$$

HOAS

usual syntax

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

$\forall x. x = 2$

Higher Order Abstract Syntax



Example:

$$\text{ALL} :: (\alpha \Rightarrow \text{bool}) \Rightarrow \text{bool}$$

HOAS

usual syntax

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

$\forall x. x = 2$

ALL P

$\forall x. P\ x$

Higher Order Abstract Syntax



Example:

$$\text{ALL} :: (\alpha \Rightarrow \text{bool}) \Rightarrow \text{bool}$$

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 in Isabelle



→ **mixfix:**

consts drvbl :: $ct \Rightarrow ct \Rightarrow fm \Rightarrow bool$ ("-, - \vdash -")

Legal syntax now: $\Gamma, \Pi \vdash F$

Side Track: Syntax Declarations in Isabelle



→ **mixfix:**

consts `drvbl :: ct ⇒ ct ⇒ fm ⇒ bool ("_,_ ⊢ _")`

Legal syntax now: $\Gamma, \Pi \vdash F$

→ **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 in Isabelle



→ **mixfix:**

consts `drvbl :: ct ⇒ ct ⇒ fm ⇒ bool ("_,_ ⊢ _")`

Legal syntax now: $\Gamma, \Pi \vdash F$

→ **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 in Isabelle



→ **mixfix:**

consts `drvbl` :: $ct \Rightarrow ct \Rightarrow fm \Rightarrow bool$ ("_,_ ⊢ _")

Legal syntax now: $\Gamma, \Pi \vdash F$

→ **priorities:**

pattern can be annotated with priorities to indicate binding strength

Example: `drvbl` :: $ct \Rightarrow ct \Rightarrow fm \Rightarrow bool$ ("_,_ ⊢ _" [30, 0, 20] 60)

→ **infixl/infixr:** short form for left/right associative binary operators

Example: `or` :: $bool \Rightarrow bool \Rightarrow bool$ (infixr " ∨ " 30)

→ **binders:** declaration must be of the form

c :: $(\tau_1 \Rightarrow \tau_2) \Rightarrow \tau_3$ (binder "B" < p >)

$B x. P$ translated into $c P$ (and vice versa)

Example `ALL` :: $(\alpha \Rightarrow bool) \Rightarrow bool$ (binder "∀" 10)

Side Track: Syntax Declarations in Isabelle



→ **mixfix:**

consts `drvbl` :: $ct \Rightarrow ct \Rightarrow fm \Rightarrow bool$ ("_,_ ⊢ _")

Legal syntax now: $\Gamma, \Pi \vdash F$

→ **priorities:**

pattern can be annotated with priorities to indicate binding strength

Example: `drvbl` :: $ct \Rightarrow ct \Rightarrow fm \Rightarrow bool$ ("_,_ ⊢ _" [30, 0, 20] 60)

→ **infixl/infixr:** short form for left/right associative binary operators

Example: `or` :: $bool \Rightarrow bool \Rightarrow bool$ (infixr " ∨ " 30)

→ **binders:** declaration must be of the form

c :: $(\tau_1 \Rightarrow \tau_2) \Rightarrow \tau_3$ (binder "B" < p >)

$B x. P$ translated into $c P$ (and vice versa)

Example `ALL` :: $(\alpha \Rightarrow bool) \Rightarrow bool$ (binder "∀" 10)

More in Isabelle/Isar Reference Manual (7.2)

Back to HOL



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

And the rest is

Back to HOL



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

And the rest is definitions:

True \equiv

All P \equiv

Ex P \equiv

False \equiv

$\neg P$ \equiv

$P \wedge Q$ \equiv

$P \vee Q$ \equiv

If $P \times y$ \equiv

inj f \equiv

surj f \equiv

Back to HOL



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

And the rest is definitions:

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

All $P \equiv$

Ex $P \equiv$

False \equiv

$\neg P \equiv$

$P \wedge Q \equiv$

$P \vee Q \equiv$

If $P \times y \equiv$

inj $f \equiv$

surj $f \equiv$

Back to HOL



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

And the rest is definitions:

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

$All\ P \quad \equiv \quad P = (\lambda x. True)$

$Ex\ P \quad \equiv$

$False \quad \equiv$

$\neg P \quad \equiv$

$P \wedge Q \quad \equiv$

$P \vee Q \quad \equiv$

$If\ P\ x\ y \quad \equiv$

$inj\ f \quad \equiv$

$surj\ f \quad \equiv$

Back to HOL



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

And the rest is definitions:

$\text{True} \quad \equiv \quad (\lambda x :: \text{bool}. x) = (\lambda x. x)$

$\text{All } P \quad \equiv \quad P = (\lambda x. \text{True})$

$\text{Ex } P \quad \equiv \quad \forall Q. (\forall x. P \ x \longrightarrow Q) \longrightarrow Q$

$\text{False} \quad \equiv \quad \forall P. P$

$\neg P \quad \equiv \quad P \longrightarrow \text{False}$

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

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

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

$\text{inj } f \quad \equiv \quad \forall x \ y. f \ x = f \ y \longrightarrow x = y$

$\text{surj } f \quad \equiv \quad \forall y. \exists x. y = f \ x$

The Axioms of HOL



$$\frac{}{t = t} \text{ refl}$$

$$\frac{s = t \quad P \ s}{P \ t} \text{ subst}$$

$$\frac{\bigwedge x. f \ x = g \ x}{(\lambda x. f \ x) = (\lambda x. g \ x)} \text{ ext}$$

The Axioms of HOL



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

The Axioms of HOL



$$\begin{array}{l} \frac{}{t = t} \text{ refl} \qquad \frac{s = t \quad P \ s}{P \ t} \text{ subst} \qquad \frac{\bigwedge x. f \ x = g \ x}{(\lambda x. f \ x) = (\lambda x. g \ x)} \text{ ext} \\ \\ \frac{P \implies Q}{P \longrightarrow Q} \text{ impl} \qquad \frac{P \longrightarrow Q \quad P}{Q} \text{ mp} \\ \\ \frac{}{(P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P = Q)} \text{ iff} \end{array}$$

The Axioms of HOL



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

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

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

$$\frac{}{P = \text{True} \vee P = \text{False}} \text{ True_or_False}$$

The Axioms of HOL



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

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

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

$$\overline{P = \text{True} \vee P = \text{False}} \text{ True_or_False}$$

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

The Axioms of HOL



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

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

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

$$\frac{}{P = \text{True} \vee P = \text{False}} \text{ True_or_False}$$

$$\frac{P \ ?_x}{P \ (\text{SOME } x. P \ x)} \text{ some1}$$

$$\frac{}{\exists f :: \text{ind} \implies \text{ind. inj } f \wedge \neg \text{surj } f} \text{ infnty}$$

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:

$$\frac{x = y}{x \equiv y} \text{ eq_reflection} \qquad \frac{}{(\text{THE } x. x = a) = a} \text{ the_eq_trivial}$$



DATA
61



Demo: The Definitions in Isabelle

Deriving Proof Rules



In the following, we will

Deriving Proof Rules



In the following, we will

→ look at the definitions in more detail

Deriving Proof Rules



In the following, we will

- look at the definitions in more detail
- derive the traditional proof rules from the axioms in Isabelle

Deriving Proof Rules



In the following, we will

- look at the definitions in more detail
- derive the traditional proof rules from the axioms in Isabelle

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

```
lemma [name :]  
assumes [name1 :] “< prop >1”  
assumes [name2 :] “< prop >2”  
⋮  
shows “< prop >” < proof >
```

Deriving Proof Rules



In the following, we will

- look at the definitions in more detail
- derive the traditional proof rules from the axioms in Isabelle

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

```
lemma [name :]  
assumes [name1 :] “< prop >1”  
assumes [name2 :] “< prop >2”  
⋮  
shows “< prop >” < proof >
```

proves: $\llbracket \langle prop \rangle_1; \langle prop \rangle_2; \dots \rrbracket \implies \langle prop \rangle$

True

consts True :: *bool*

True $\equiv (\lambda x :: \textit{bool}. x) = (\lambda x. x)$

Intuition:

right hand side is always true



True



consts True :: *bool*

True \equiv ($\lambda x :: \textit{bool}. x$) = ($\lambda x. x$)

Intuition:

right hand side is always true

Proof Rules:

$$\overline{\text{True}} \text{ TrueI}$$

Proof:

$$\frac{(\lambda x :: \textit{bool}. x) = (\lambda x. x)}{\text{True}} \begin{array}{l} \text{refl} \\ \text{unfold True_def} \end{array}$$

A background pattern of white hexagons on a dark teal background, arranged in a staggered grid.

DATA
61



Demo

Universal Quantifier

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



Universal Quantifier



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

Intuition:

→ ALL P is Higher Order Abstract Syntax for $\forall x. P x$.

Universal Quantifier



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

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.

Universal Quantifier



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

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 $\lambda x. \text{True}$.

Universal Quantifier



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

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 $\lambda x. \text{True}$.

Proof Rules:

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

Proof: Isabelle Demo

False

consts False :: *bool*
False $\equiv \forall P.P$



False

consts False :: *bool*

False $\equiv \forall P.P$

Intuition:

Everything can be derived from *False*.



False



consts False :: *bool*

False $\equiv \forall P.P$

Intuition:

Everything can be derived from *False*.

Proof Rules:

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

Proof: Isabelle Demo

Negation

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



Negation



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

Intuition:

Try $P = \text{True}$ and $P = \text{False}$ and the traditional truth table for \longrightarrow .

Negation



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

Intuition:

Try $P = \text{True}$ and $P = \text{False}$ and the traditional truth table for \longrightarrow .

Proof Rules:

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

Proof: Isabelle Demo

Existential Quantifier



consts EX :: ($\alpha \Rightarrow \text{bool}$) \Rightarrow *bool*
EX P $\equiv \forall Q. (\forall x. P\ x \longrightarrow Q) \longrightarrow Q$

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)

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

Existential Quantifier



consts EX :: ($\alpha \Rightarrow \text{bool}$) \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: $(\forall x. P\ x \longrightarrow Q)$

Existential Quantifier



consts EX :: ($\alpha \Rightarrow \text{bool}$) \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: $(\forall x. P\ x \longrightarrow Q) = ((\exists x. P\ x) \longrightarrow Q)$

Existential Quantifier



consts EX :: ($\alpha \Rightarrow \text{bool}$) \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* ($- \wedge -$)
 $P \wedge Q \equiv \forall R. (P \longrightarrow Q \longrightarrow R) \longrightarrow R$

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

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

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 -)

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

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)

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

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:

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

Proof: Isabelle Demo

If-Then-Else



consts If $:: \text{bool} \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \alpha$ (if_ then _ else _)

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

If-Then-Else



consts If $:: \text{bool} \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \alpha$ (if_ then _ else _)

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

Intuition:

→ for $P = \text{True}$, right hand side collapses to $\text{SOME } z. z = x$

If-Then-Else



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

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

Intuition:

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

If-Then-Else



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

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

Intuition:

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

Proof Rules:

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

Proof: Isabelle Demo



DATA
61



That was HOL

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

This can be automated

More on Automation



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

This can be automated

More on Automation



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

This can be automated

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

This can be automated

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

This can be automated

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

remove attribute globally

use locally

delete locally

declare conjI [intro!] allE [elim]

declare allE [rule del]

apply (blast intro: someI)

apply (blast del: conjI)

A white hexagonal grid pattern is overlaid on a teal background, covering the top half of the slide.

DATA
61



Demo: Automation

We have learned today ...



→ Defining HOL

We have learned today ...



- Defining HOL
- Higher Order Abstract Syntax

We have learned today ...



- Defining HOL
- Higher Order Abstract Syntax
- Deriving proof rules

We have learned today ...



- Defining HOL
- Higher Order Abstract Syntax
- Deriving proof rules
- More automation