Script **generated by TTT**

Title: FDS (04.05.2018)

Date: Fri May 04 08:31:28 CEST 2018

Duration: 86:06 min

Pages: 96

Chapter 4

Logic and Proof Beyond Equality

maximize

Syntax (in decreasing precedence):

$$
\begin{array}{lllllll}
form & ::= & (form) & \mid & term = term & \mid & \neg form \\
 & \mid & form \wedge form & \mid & form \vee form & \mid & form \longrightarrow form \\
 & \mid & \forall x.\ form & \mid & \exists x.\ form & &
\end{array}
$$

Syntax (in decreasing precedence):

$$\begin{array}{lclclcl} form & ::= & (form) & \mid & term = term & \mid & \neg form \\ & \mid & form \wedge form & \mid & form \vee form & \mid & form \longrightarrow form \\ & \mid & \forall x.\ form & \mid & \exists x.\ form & & \end{array}$$

Examples:

$$\neg\ A \wedge B \vee C \quad \equiv \quad ((\neg\ A) \wedge B) \vee C$$

Syntax (in decreasing precedence):

$$\begin{array}{lclclcl} form & ::= & (form) & \mid & term = term & \mid & \neg form \\ & \mid & form \wedge form & \mid & form \vee form & \mid & form \longrightarrow form \\ & \mid & \forall x.\ form & \mid & \exists x.\ form & & \end{array}$$

Examples:

$$\begin{array}{rcl} \neg\ A \wedge B \vee C & \equiv & ((\neg\ A) \wedge B) \vee C \\ s = t \wedge C & \equiv & (s = t) \wedge C \end{array}$$

Syntax (in decreasing precedence):

$$\begin{array}{lclclcl} form & ::= & (form) & \mid & term = term & \mid & \neg form \\ & \mid & form \wedge form & \mid & form \vee form & \mid & form \longrightarrow form \\ & \mid & \forall x.\ form & \mid & \exists x.\ form & & \end{array}$$

Examples:

$$\begin{array}{rcl} \neg\ A \wedge B \vee C & \equiv & ((\neg\ A) \wedge B) \vee C \\ s = t \wedge C & \equiv & (s = t) \wedge C \\ A \wedge B = B \wedge A & \equiv & \color{red}{A \wedge (B = B) \wedge A} \end{array}$$

Syntax (in decreasing precedence):

$$\begin{array}{lclclcl} form & ::= & (form) & \mid & term = term & \mid & \neg form \\ & \mid & form \wedge form & \mid & form \vee form & \mid & form \longrightarrow form \\ & \mid & \forall x.\ form & \mid & \exists x.\ form & & \end{array}$$

Examples:

$$\begin{array}{rcl} \neg\ A \wedge B \vee C & \equiv & ((\neg\ A) \wedge B) \vee C \\ s = t \wedge C & \equiv & (s = t) \wedge C \\ A \wedge B = B \wedge A & \equiv & \color{red}{A \wedge (B = B) \wedge A} \\ \forall\, x.\ P\ x \wedge Q\ x & \equiv & \forall\, x.\ (P\ x \wedge Q\ x) \end{array}$$

Syntax (in decreasing precedence):

$$
\begin{array}{lllll}
\mathit{form} ::= & (\mathit{form}) & \mid\ \mathit{term} = \mathit{term} & \mid\ \neg \mathit{form} \\
\mid & \mathit{form} \wedge \mathit{form} & \mid\ \mathit{form} \vee \mathit{form} & \mid\ \mathit{form} \longrightarrow \mathit{form} \\
\mid & \forall x.\ \mathit{form} & \mid\ \exists x.\ \mathit{form} &
\end{array}
$$

Examples:

$$
\begin{array}{rcl}
\neg\ A \wedge B \vee C & \equiv & ((\neg\ A) \wedge B) \vee C \\
s = t \wedge C & \equiv & (s = t) \wedge C \\
A \wedge B = B \wedge A & \equiv & A \wedge (B = B) \wedge A \\
\forall\, x.\ P\ x \wedge Q\ x & \equiv & \forall\, x.\ (P\ x \wedge Q\ x)
\end{array}
$$

Input syntax: $\longleftrightarrow$ (same precedence as $\longrightarrow$)

Variable binding convention:

$$\forall\, x\ y.\ P\ x\ y \ \equiv\ \forall\, x.\ \forall\, y.\ P\ x\ y$$

Warning

Quantifiers have low precedence
and need to be parenthesized (if in some context)

! $P \wedge \forall\, x.\ Q\ x \ \leadsto\ P \wedge (\forall\, x.\ Q\ x)$ **!**

Mathematical symbols

. . . and their ascii representations:

$\forall$	`\<forall>`	`ALL`
$\exists$	`\<exists>`	`EX`
λ	`\<lambda>`	`%`
$\longrightarrow$	`-->`	
$\longleftrightarrow$	`<->`	
$\wedge$	`/\`	`&`
$\vee$	`\/`	`|`
$\neg$	`\<not>`	`~`
$\neq$	`\<noteq>`	`~=`

Sets over type $'a$

$'a\ set$

Sets over type $'a$

$'a\ set$

- $\{\}, \quad \{e_1,\ldots,e_n\}$

Sets over type $'a$

$'a\ set$

- $\{\}, \quad \{e_1,\ldots,e_n\}$
- $e \in A, \quad A \subseteq B$
- $A \cup B, \quad A \cap B, \quad A - B, \quad - A$

Sets over type $'a$

$'a\ set$

- $\{\}, \quad \{e_1,\ldots,e_n\}$
- $e \in A, \quad A \subseteq B$
- $A \cup B, \quad A \cap B, \quad A - B, \quad - A$
- $\{x.\ P\}$ where x is a variable

Sets over type $'a$

$'a$ *set*

- $\{\}$, $\{e_1,\ldots,e_n\}$
- $e \in A$, $A \subseteq B$
- $A \cup B$, $A \cap B$, $A - B$, $- A$
- $\{x.\ P\}$ where x is a variable
- $\ldots$

Sets over type $'a$

$'a$ *set*

- $\{\}$, $\{e_1,\ldots,e_n\}$
- $e \in A$, $A \subseteq B$
- $A \cup B$, $A \cap B$, $A - B$, $- A$
- $\{x.\ P\}$ where x is a variable
- $\ldots$

$\in$	`\<in>`	`:`
$\subseteq$	`\<subseteq>`	`<=`
$\cup$	`\<union>`	`Un`
$\cap$	`\<inter>`	`Int`

5 Logical Formulas

6 Proof Automation

7 Single Step Proofs

simp and *auto*

simp: rewriting and a bit of arithmetic

auto: rewriting and a bit of arithmetic, logic and sets

simp and *auto*

simp: rewriting and a bit of arithmetic

auto: rewriting and a bit of arithmetic, logic and sets

- Show you where they got stuck

simp and *auto*

simp: rewriting and a bit of arithmetic

auto: rewriting and a bit of arithmetic, logic and sets

- Show you where they got stuck
- highly incomplete
- Extensible with new *simp*-rules

Exception: *auto* acts on all subgoals

fastforce

- rewriting, logic, sets, relations and a bit of arithmetic.

fastforce

- rewriting, logic, sets, relations and a bit of arithmetic.
- incomplete but better than *auto*.
- Succeeds or fails

blast

- A complete proof search procedure for FOL . . .

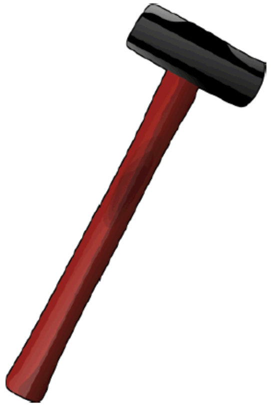
blast

- A complete proof search procedure for FOL . . .
- . . . but (almost) without "="

blast

- A complete proof search procedure for FOL . . .
- . . . but (almost) without "="
- Covers logic, sets and relations

Sledgehammer



chitecture:

Isabelle

external
ATPs[1]

[1] Automatic Theorem Provers

chitecture:

Isabelle

Goal
& filtered library ↓

external
ATPs[1]

[1] Automatic Theorem Provers

chitecture:

Isabelle

Goal
& filtered library ↓ ↑ Proof

external
ATPs[1]

[1] Automatic Theorem Provers

Architecture:

Isabelle

$$\begin{array}{ccc} \text{Goal} & & \\ \text{\& filtered library} & \downarrow \;\; \uparrow & \text{Proof} \end{array}$$

external
ATPs[1]

Characteristics:

- Sometimes it works,
- sometimes it doesn't.

[1] Automatic Theorem Provers

by$(proof\text{-}method)$

$\approx$

apply$(proof\text{-}method)$
done

`Auto_Proof_Demo.thy`

Linear formulas

Linear formulas

Only:

variables

numbers

number $*$ variable

Linear formulas

Only:

variables

numbers

number $*$ variable

$+, -$

111

Linear formulas

Only:

variables

numbers

number $*$ variable

$+, -$

$=, \leq, <$

$\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$

111

Linear formulas

Only:

variables

numbers

number $*$ variable

$+, -$

$=, \leq, <$

$\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$

Examples

Linear: $3 * x + 5 * y \leq z \longrightarrow x < z$

Linear formulas

Only:

variables

numbers

number $*$ variable

$+, -$

$=, \leq, <$

$\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$

Examples

Linear: $3 * x + 5 * y \leq z \longrightarrow x < z$

Nonlinear: $x \leq x * x$

Extended linear formulas

Also allowed:

min, max

$even$, odd

$t\ div\ n$, $t\ mod\ n$ where n is a number

conversion functions

nat, $floor$, $ceiling$, abs

Automatic proof of arithmetic formulas

by *arith*

Automatic proof of arithmetic formulas

by *arith*

Proof method *arith* tries to prove arithmetic formulas.

- Succeeds or fails
- Decision procedure for extended linear formulas

Automatic proof of arithmetic formulas

by *arith*

Proof method *arith* tries to prove arithmetic formulas.

- Succeeds or fails
- Decision procedure for extended linear formulas
- Nonlinear subterms are viewed as (new) variables.
  Example: $x \leq x * x + f\ y$ is viewed as $x \leq u + v$

Automatic proof of arithmetic formulas

by (*simp add: algebra_simps*)

Automatic proof of arithmetic formulas

by (*simp add: algebra_simps*)

- The lemmas list *algebra_simps* helps to simplify arithmetic formulas
- It contains associativity, commutativity and distributivity of $+$ and $*$.

Automatic proof of arithmetic formulas

by (*simp add: algebra_simps*)

- The lemmas list *algebra_simps* helps to simplify arithmetic formulas
- It contains associativity, commutativity and distributivity of $+$ and $*$.
- This may prove the formula, may make it simpler, or may make it unreadable.

Automatic proof of arithmetic formulas

by (*simp add: field_simps*)

Automatic proof of arithmetic formulas

by (*simp add: field_simps*)

- The lemmas list *field_simps* extends *algebra_simps* by rules for /

Automatic proof of arithmetic formulas

by (*simp add: field_simps*)

- The lemmas list *field_simps* extends *algebra_simps* by rules for /
- Can only cancel common terms in a quotient, e.g. $x * y \ / \ (x * z)$,

Automatic proof of arithmetic formulas

by (*simp add: field_simps*)

- The lemmas list *field_simps* extends *algebra_simps* by rules for /
- Can only cancel common terms in a quotient, e.g. $x * y \ / \ (x * z)$, if $x \neq 0$ can be proved.

Numerals

Numerals are syntactically different from Suc-terms.

Numerals

Numerals are syntactically different from Suc-terms.
Therefore numerals do not match Suc-patterns.

Numerals

Numerals are syntactically different from Suc-terms.
Therefore numerals do not match Suc-patterns.

Example

Exponentiation $x\ \hat{}\ n$ is defined by Suc-recursion on n.

Numerals

Numerals are syntactically different from Suc-terms.
Therefore numerals do not match Suc-patterns.

Example

Exponentiation $x\ \hat{}\ n$ is defined by Suc-recursion on n.
Therefore $x\ \hat{}\ 2$ is not simplified by $simp$ and $auto$.

Numerals

Numerals are syntactically different from *Suc*-terms.
Therefore numerals do not match *Suc*-patterns.

Example

Exponentiation $x \,\hat{}\, n$ is defined by *Suc*-recursion on n.
Therefore $x \,\hat{}\, 2$ is not simplified by *simp* and *auto*.

Numerals can be converted into *Suc*-terms with rule *numeral_eq_Suc*

Numerals

Numerals are syntactically different from *Suc*-terms.
Therefore numerals do not match *Suc*-patterns.

Example

Exponentiation $x \,\hat{}\, n$ is defined by *Suc*-recursion on n.
Therefore $x \,\hat{}\, 2$ is not simplified by *simp* and *auto*.

Numerals can be converted into *Suc*-terms with rule *numeral_eq_Suc*

Example

simp add: numeral_eq_Suc rewrites $x \,\hat{}\, 2$ to $x * x$

`Auto_Proof_Demo.thy`

Arithmetic

Step-by-step proofs can be necessary if automation fails and you have to explore where and why it failed by taking the goal apart.

What are these *?-variables* ?

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\leadsto$

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\leadsto$
  $[\![a = b; False]\!] \Longrightarrow a = b \wedge False$
- By unification:
  unifying $?P \wedge ?Q$ with $a{=}b \wedge False$
  sets $?P$ to $a{=}b$ and $?Q$ to $False$.

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\leadsto$
  $[\![a = b; False]\!] \Longrightarrow a = b \wedge False$

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\leadsto$
  $[\![a = b; False]\!] \Longrightarrow a = b \wedge False$
- By unification:
  unifying $?P \wedge ?Q$ with $a{=}b \wedge False$

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\rightsquigarrow$
  $[\![a = b; False]\!] \Longrightarrow a = b \wedge False$
- By unification:
  unifying $?P \wedge ?Q$ with $a{=}b \wedge False$
  sets $?P$ to $a{=}b$ and $?Q$ to $False$.

Rule application

Rule application

Example: rule: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$
subgoal: $1.\ \ldots \Longrightarrow A \wedge B$

Rule application

Example: rule: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$
subgoal: $1.\ \ldots \Longrightarrow A \wedge B$

Result: $1.\ \ldots \Longrightarrow A$
$2.\ \ldots \Longrightarrow B$

Rule application

Example: rule: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$
subgoal: 1. $\ldots \Longrightarrow A \wedge B$

Result: 1. $\ldots \Longrightarrow A$
2. $\ldots \Longrightarrow B$

The general case: applying rule $[\![\ A_1; \ldots ; A_n\ ]\!] \Longrightarrow A$ to subgoal $\ldots \Longrightarrow C$:

Rule application

Example: rule: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$
subgoal: 1. $\ldots \Longrightarrow A \wedge B$

Result: 1. $\ldots \Longrightarrow A$
2. $\ldots \Longrightarrow B$

The general case: applying rule $[\![\ A_1; \ldots ; A_n\ ]\!] \Longrightarrow A$ to subgoal $\ldots \Longrightarrow C$:

- Unify A and C

Rule application

Example: rule: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$
subgoal: 1. $\ldots \Longrightarrow A \wedge B$

Result: 1. $\ldots \Longrightarrow A$
2. $\ldots \Longrightarrow B$

The general case: applying rule $[\![\ A_1; \ldots ; A_n\ ]\!] \Longrightarrow A$ to subgoal $\ldots \Longrightarrow C$:

- Unify A and C

- Replace C with n new subgoals $A_1 \ldots A_n$

Rule application

Example: rule: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$
subgoal: 1. $\ldots \Longrightarrow A \wedge B$

Result: 1. $\ldots \Longrightarrow A$
2. $\ldots \Longrightarrow B$

The general case: applying rule $[\![\ A_1; \ldots ; A_n\ ]\!] \Longrightarrow A$ to subgoal $\ldots \Longrightarrow C$:

- Unify A and C
- Replace C with n new subgoals $A_1 \ldots A_n$

apply$(\mathit{rule}\ xyz)$

Rule application

Example: rule: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$
subgoal: $1.\ \ldots \Longrightarrow A \wedge B$

Result: $1.\ \ldots \Longrightarrow A$
$2.\ \ldots \Longrightarrow B$

The general case: applying rule $[\![\ A_1; \ldots ; A_n\ ]\!] \Longrightarrow A$ to subgoal $\ldots \Longrightarrow C$:

- Unify A and C
- Replace C with n new subgoals $A_1 \ldots A_n$

apply($rule\ xyz$)

"Backchaining"

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \wedge ?Q}\ \texttt{conjI}$$

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \wedge ?Q}\ \texttt{conjI}$$

$$\frac{?P \Longrightarrow ?Q}{?P \longrightarrow ?Q}\ \texttt{impI}$$

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \wedge ?Q}\ \texttt{conjI}$$

$$\frac{?P \Longrightarrow ?Q}{?P \longrightarrow ?Q}\ \texttt{impI}$$

$$\frac{\bigwedge x.\ ?P\ x}{\forall x.\ ?P\ x}\ \texttt{allI}$$

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \wedge ?Q}\ \texttt{conjI}$$

$$\frac{?P \Longrightarrow ?Q}{?P \longrightarrow ?Q}\ \texttt{impI} \qquad \frac{\bigwedge x.\ ?P\ x}{\forall x.\ ?P\ x}\ \texttt{allI}$$

$$\frac{?P \Longrightarrow ?Q \quad ?Q \Longrightarrow ?P}{?P = ?Q}\ \texttt{iffI}$$

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \wedge ?Q}\ \texttt{conjI}$$

$$\frac{?P \Longrightarrow ?Q}{?P \longrightarrow ?Q}\ \texttt{impI} \qquad \frac{\bigwedge x.\ ?P\ x}{\forall x.\ ?P\ x}\ \texttt{allI}$$

$$\frac{?P \Longrightarrow ?Q \quad ?Q \Longrightarrow ?P}{?P = ?Q}\ \texttt{iffI}$$

They are known as introduction rules
because they *introduce* a particular connective.

Forward proof: OF

If r is a theorem $A \Longrightarrow B$

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \wedge ?Q}\ \texttt{conjI}$$

$$\frac{?P \Longrightarrow ?Q}{?P \longrightarrow ?Q}\ \texttt{impI}$$

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\leadsto$
  $[\![a = b; False]\!] \Longrightarrow a = b \wedge False$
- By unification:
  unifying $?P \wedge ?Q$ with $a{=}b \wedge False$
  sets $?P$ to $a{=}b$ and $?Q$ to $False$.

Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A

Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A then

$$r[OF\ s]$$

is the theorem obtained by proving A with s.

Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A then

$$r[OF\ s]$$

is the theorem obtained by proving A with s.

Example: theorem `refl`: $?t = ?t$

Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A then

$$r[OF\ s]$$

is the theorem obtained by proving A with s.

Example: theorem `refl`: $?t = ?t$

```
conjI[OF refl[of "a"]]
```


The general case:

If r is a theorem $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$
and $r_1, \ldots, r_m$ $(m \leq n)$ are theorems then

$$r[OF\ r_1\ \ldots\ r_m]$$

is the theorem obtained
by proving $A_1 \ldots A_m$ with $r_1 \ldots r_m$.

The general case:

If r is a theorem $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$
and $r_1, \ldots, r_m$ $(m \leq n)$ are theorems then

$$r[OF\ r_1\ \ldots\ r_m]$$

is the theorem obtained
by proving $A_1 \ldots A_m$ with $r_1 \ldots r_m$.

Example: theorem `refl`: $?t = ?t$

```
conjI[OF refl[of "a"] refl[of "b"]]
```


From now on: $?$ mostly suppressed on slides

Single_Step_Demo.thy

$\Longrightarrow$ versus $\longrightarrow$

$\Longrightarrow$ is part of the Isabelle framework. It structures theorems and proof states: $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$

$\longrightarrow$ is part of HOL and can occur inside the logical formulas A_i and A.

$\Longrightarrow$ versus $\longrightarrow$

$\Longrightarrow$ is part of the Isabelle framework. It structures theorems and proof states: $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$

$\longrightarrow$ is part of HOL and can occur inside the logical formulas A_i and A.

Phrase theorems like this $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$
not like this $A_1 \wedge \ldots \wedge A_n \longrightarrow A$

$\Longrightarrow$ versus $\longrightarrow$

$\Longrightarrow$ is part of the Isabelle framework. It structures theorems and proof states: $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$

$\longrightarrow$ is part of HOL and can occur inside the logical formulas A_i and A.

Phrase theorems like this $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$
not like this $A_1 \wedge \ldots \wedge A_n \longrightarrow A$

Bookmarks

$\Longrightarrow$ versus $\longrightarrow$

$\Longrightarrow$ is part of the Isabelle framework. It structures theorems and proof states: $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$

$\longrightarrow$ is part of HOL and can occur inside the logical formulas A_i and A.

Phrase theorems like this $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$
not like this $A_1 \wedge \ldots \wedge A_n \longrightarrow A$