Chapter 4

Logic and Proof Beyond Equality

Syntax (in decreasing precedence):

\[
\begin{align*}
\text{form} & ::= \text{term} \quad \text{term} = \text{term} \\
& \quad \neg \text{form} \\
& \quad \forall x. \text{form} \\
& \quad \exists x. \text{form} \\
& \quad (\text{form} \land \text{form}) \\
& \quad \text{form} \lor \text{form} \\
\end{align*}
\]
Syntax (in decreasing precedence):

\[
\begin{align*}
\text{form} &::= \text{(form)} \quad | \quad \text{term} = \text{term} \quad | \quad \neg \text{form} \\
&| \quad \text{form} \land \text{form} \quad | \quad \text{form} \lor \text{form} \quad | \quad \text{form} \rightarrow \text{form} \\
&| \quad \forall x. \text{form} \quad | \quad \exists x. \text{form}
\end{align*}
\]

Examples:

\[
\begin{align*}
\neg A \land B \lor C &\equiv ((\neg A) \land B) \lor C \\
s = t \land C &\equiv (s = t) \land C \\
A \land B = B \land A &\equiv A \land (B = B) \land A
\end{align*}
\]
Syntax (in decreasing precedence):

\[
\begin{align*}
\text{form} & ::= (\text{form}) \\
& \mid \text{form} \land \text{form} \\
& \mid \forall x. \text{form} \\
& \mid \text{term} = \text{term} \\
& \mid \neg \text{form} \\
& \mid \text{form} \rightarrow \text{form}
\end{align*}
\]

Examples:

\[
\begin{align*}
\neg A \land B \lor C & \equiv ((\neg A) \land B) \lor C \\
s = t \land C & \equiv (s = t) \land C \\
A \land B = B \land A & \equiv A \land (B = B) \land A \\
\forall x. P x \land Q x & \equiv \forall x. (P x \land Q x)
\end{align*}
\]

Input syntax: \(\leftarrow\) (same precedence as \(\rightarrow\))

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)

\[
\neg P \land \forall x. Q x \equiv \neg P \land (\forall x. Q x)
\]

Mathematical symbols

\[
\begin{align*}
\forall & \ \langle\text{forall}\rangle \ \text{ALL} \\
\exists & \ \langle\text{exists}\rangle \ \text{EX} \\
\\lambda & \ \langle\text{lambda}\rangle \ \% \\
\rightarrow & \ \rightarrow\rightarrow \\
\leftrightarrow & \ \leftrightarrow\leftrightarrow \\
\& & \ \& \\
\mid & \ \mid \\
\neg & \ \langle\text{not}\rangle \ \sim \\
\not= & \ \langle\text{not}\text{eq}\rangle \ \sim= 
\end{align*}
\]
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

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
Sets over type 'a

'a set

- {}, \{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
- ...
**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 "="
- Covers logic, sets and relations
Sledgehammer

Architecture:

Isabelle

external ATPs

1Automatic Theorem Provers

Architecture:

Isabelle

Goal & filtered library

external ATPs

1Automatic Theorem Provers

Architecture:

Isabelle

Goal & filtered library

external ATPs

Proof

1Automatic Theorem Provers
Architecture:

**Isabelle**

Goal & filtered library

\[ \downarrow \uparrow \]

Proof

external ATPs\(^1\)

Characteristics:

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

\(^1\)Automatic Theorem Provers

by (proof-method)

\[ \approx \]

apply (proof-method)

done

Linear formulas

Auto_Proof_Demo.thy
Linear formulas

Only:
  variables
  numbers
  number * variable
  +, −
  =, ≤, <
  ¬, ∧, ∨, →, ↔

Examples
Linear: 3 * x + 5 * y ≤ z → x < z
Linear formulas

Only:

variables
numbers
number \ast variable
+,
−
=, \leq, <
\neg, \land, \lor, \rightarrow, \leftrightarrow

Examples

Linear: \quad 3 \ast x + 5 \ast y \leq z \rightarrow x < z
Nonlinear: \quad x \leq x \ast 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 \textit{arith}

Proof method \textit{arith} tries to prove arithmetic formulas.

- Succeeds or fails
- Decision procedure for extended linear formulas

Automatic proof of arithmetic formulas

by \textit{arith}
Proof method \textit{arith} tries to prove arithmetic formulas.
- Succeeds or fails
- Decision procedure for extended linear formulas
- Nonlinear subterms are viewed as (new) variables.
  Example: $x \leq x \cdot x + f \cdot y$ is viewed as $x \leq u + v$

- The lemmas list \textit{algebra_simps} helps to simplify arithmetic formulas
- It contains associativity, commutativity and distributivity of $+$ and $\cdot$.

- The lemmas list \textit{algebra_simps} helps to simplify arithmetic formulas
- It contains associativity, commutativity and distributivity of $+$ and $\cdot$.
- This may prove the formula, may make it simpler, or may make it unreadable.
Automatic proof of arithmetic formulas
by (simp add: field_simsps)

- The lemmas list *field_simsps* extends *algebra_simsps*
  by rules for `/`

- Can only cancel common terms in a quotient,
  e.g. `x * y / (x * z)`, if `x ≠ 0` can be proved.
Numerals

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

Example
Exponentiation $x ^ n$ is defined by $Suc$-recursion on $n$. Therefore $x ^ 2$ is not simplified by simp and auto.
Numerals

Numerals are syntactically different from $\text{Suc}$-terms. Therefore numerals do not match $\text{Suc}$-patterns.

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

Numerals can be converted into $\text{Suc}$-terms with rule \texttt{numeral_eq_Suc}

Numerals

Numerals are syntactically different from $\text{Suc}$-terms. Therefore numerals do not match $\text{Suc}$-patterns.

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

Numerals can be converted into $\text{Suc}$-terms with rule \texttt{numeral_eq_Suc}

Example
\texttt{simp add: numeral_eq_Suc} rewrites $x \cdot 2$ to $x \cdot x$

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?

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


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: $[?P; ?Q] \Longrightarrow ?P \land ?Q$

These $?$-variables can later be instantiated:

- By hand:
  \[
  \text{conjI[of "a=b" "False"]} \leadsto \\
  [a = b; False] \Longrightarrow a = b \land False
  \]

- By unification:
  unifying $?P \land ?Q$ with $a=b \land False$ 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: \([?-P; \ ?Q] \rightarrow \ ?P \land \ ?Q\)

These ?-variables can later be instantiated:

- By hand:
  \[\text{conjI[of "a=b" "False"]} \leadsto \[a = b; \ False]\rightarrow a = b \land False\]

- By unification:
  unifying \(?P \land \ ?Q\) with \(a=b \land False\)
  sets \(?P\) to \(a=b\) and \(?Q\) to \(False\).

Rule application

Example: rule: \([?-P; \ ?Q] \rightarrow \ ?P \land \ ?Q\)
subgoal: 1. \(\ldots \rightarrow A \land B\)

Rule application

Example: rule: \([?-P; \ ?Q] \rightarrow \ ?P \land \ ?Q\)
subgoal: 1. \(\ldots \rightarrow A \land B\)
Result: 1. \(\ldots \rightarrow A\)
2. \(\ldots \rightarrow B\)
Rule application

Example: \[ \text{rule: } [?P; ?Q] \rightarrow ?P \land ?Q \]

subgoal: 1. \[ \ldots \rightarrow A \land B \]

Result: 1. \[ \ldots \rightarrow A \]
2. \[ \ldots \rightarrow B \]

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

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

---

Rule application

Example: \[ \text{rule: } [?P; ?Q] \rightarrow ?P \land ?Q \]

subgoal: 1. \[ \ldots \rightarrow A \land B \]

Result: 1. \[ \ldots \rightarrow A \]
2. \[ \ldots \rightarrow B \]

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

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

\textbf{apply}(rule \textit{xyz})
Rule application

Example: rule: $[?P; ?Q] \rightarrow ?P \land ?Q$

subgoal: 1. $\ldots \rightarrow A \land B$

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

The general case: applying rule $[ A_1; \ldots ; A_n ] \rightarrow A$
to subgoal $\ldots \rightarrow C$:
- Unify $A$ and $C$
- Replace $C$ with $n$ new subgoals $A_1 \ldots A_n$

apply(rule xyz)

"Backchaining"

Typical backwards rules

$\frac{?P \quad ?Q}{?P \land ?Q}$ conjI

$\frac{?P \rightarrow ?Q}{?P \rightarrow ?Q}$ impI

Typical backwards rules

$\frac{?P \rightarrow ?Q}{?P \rightarrow ?Q}$ impI

$\frac{\forall x. ?P x}{?P x}$ allI
Typical backwards rules

\[ \frac{?P \quad ?Q}{?P \land ?Q} \text{ conjI} \]

\[ \frac{?P \implies ?Q \quad \forall x. ?P x}{?P \implies ?Q} \text{ impI} \]

\[ \frac{?P \implies ?Q \quad ?Q \implies ?P}{?P = ?Q} \text{ iffI} \]

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

Forward proof: OF

If \( r \) is a theorem \( A \implies B \)
What are these \textit{?}-variables?

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

Example: \texttt{theorem conjI: }$[?P; ?Q] \Rightarrow ?P \land ?Q$

These $?\text{-variables}$ can later be instantiated:

- By hand:
  
  \texttt{conjI[of "a=b" "False"] \Rightarrow}
  
  $[a = b; False] \Rightarrow a = b \land False$

- By \texttt{unification}:
  
  unifying $?P \land ?Q$ with $a = b \land False$
  
  sets $?P$ to $a = b$ and $?Q$ to $False$.

\section*{Forward proof: OF}

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

\[ r[OF\ s] \]

is the theorem obtained by proving $A$ with $s$.

\section*{Forward proof: OF}

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

\[ r[OF\ s] \]

is the theorem obtained by proving $A$ with $s$.

Example: \texttt{theorem refl: }$?t = ?t$
Forward proof: OF

If \( r \) is a theorem \( A \implies B \) and \( s \) is a theorem that unifies with \( A \) then

\[ r[\text{OF } s] \]

is the theorem obtained by proving \( A \) with \( s \).

Example: theorem refl: \( ?t = ?t \)

\[ \text{conjI[OF refl[of } "a"\text{]]} \]

The general case:

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

\[ r[\text{OF } r_1 \ldots r_m] \]

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

Example: theorem refl: \( ?t = ?t \)

\[ \text{conjI[OF refl[of } "a"\text{]} refl[of } "b"\text{]} \]

From now on: ? mostly suppressed on slides
Single Step Demo.thy

is part of the Isabelle framework. It structures theorems and proof states: \([ A_1; \ldots; A_n ] \Rightarrow A\)

is part of HOL and can occur inside the logical formulas \(A_i\) and \(A\).

Phrase theorems like this \([ A_1; \ldots; A_n ] \Rightarrow A\)
not like this \(A_1 \land \ldots \land A_n \Rightarrow A\)
is part of the Isabelle framework. It structures theorems and proof states: \[ A_1; \ldots; A_n \Rightarrow A \]

is part of HOL and can occur inside the logical formulas \( A_i \) and \( A \).

Phrase theorems like this \[ A_1; \ldots; A_n \Rightarrow A \]
not like this \( A_1 \land \ldots \land A_n \Rightarrow A \)