Chapter 4

Logic and Proof
Beyond Equality

Syntax (in decreasing precedence):

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

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

Examples:

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

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

Examples:

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

Input syntax: $\longleftrightarrow$ (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)

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

---

**Mathematical symbols**

... and their ascii representations:

\[
\begin{align*}
\forall & \quad \texttt{\{forall\}} & \text{ALL} \\
\exists & \quad \texttt{\{exists\}} & \text{EX} \\
\lambda & \quad \texttt{\{lambda\}} & \% \\
\rightarrow & \quad \texttt{--->} & | \\
\leftrightarrow & \quad \texttt{<=-} & | \\
\land & \quad \texttt{\&} & | \\
\lor & \quad \texttt{\|} & | \\
\neg & \quad \texttt{\{not\}} & ~ \\
\neq & \quad \texttt{\{noteq\}} & ~=
\end{align*}
\]
Sets over type 'a

'a set

- $\{\}, \{e_1, \ldots, e_n\}$
- $e \in A, \ A \subseteq B$
- $A \cup B, \ A \cap B, \ A - B, \ - A$
- $\{x.\ P\}$ 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
• ...

\in \quad \text{\texttt{\textless in\textgreater}}
\subseteq \quad \text{\texttt{\textless subseteq\textgreater}} \quad \textless\textless\
\cup \quad \text{\texttt{\textless union\textgreater}} \quad \text{\texttt{Un}}
\cap \quad \text{\texttt{\textless inter\textgreater}} \quad \text{\texttt{Int}}

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

Exception: auto acts on all subgoals

fastforce

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

- incomplete but better than auto.
- Succeeds or fails
A complete proof search procedure for FOL...

...but (almost) without "="

Covers logic, sets and relations
Succeeds or fails
1Automatic Theorem Provers

1Automatic Theorem Provers

1Automatic Theorem Provers

1Automatic Theorem Provers

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

Do you feel lucky?
by (proof-method)

≈

apply (proof-method)
done

Proof Automation
Automating Arithmetic

Linear formulas
Linear formulas

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

Examples
Linear: \[ 3x + 5y \leq z \rightarrow x < z \]
Automatic proof of arithmetic formulas
by arith

Proof method \textit{arith} tries to prove arithmetic formulas.
\begin{itemize}
\item Succeeds or fails
\item Decision procedure for extended linear formulas
\item Nonlinear subterms are viewed as (new) variables.
\end{itemize}
Example: \( x \leq x \ast x + f y \) is viewed as \( x \leq u + v \)
Automatic proof of arithmetic formulas
by \((\text{simp add: algebra\_simps})\)

- The lemmas list \texttt{algebra\_simps} helps to simplify arithmetic formulas
- The lemmas list \texttt{algebra\_simps} helps to simplify arithmetic formulas
- It contains associativity, commutativity and distributivity of + and *.

Automatic proof of arithmetic formulas
by \((\text{simp add: field\_simps})\)

- The lemmas list \texttt{field\_simps} extends \texttt{algebra\_simps}
  by rules for /
Automatic proof of arithmetic formulas
by \((\text{simp add: field_simps})\)

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

---

\[114\]

Numerals

Numerals are syntactically different from \texttt{Suc}-terms.

---

\[115\]

Numerals

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

Numerals can be converted into Suc-terms with rule \texttt{numeral_eq Suc}

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

\texttt{Auto_Proof_Demo.thy}

\texttt{simp add: numeral_eqSuc rewrites $x^2$ to $x \times x$}
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:
  
  ```
  conjI[of "a=b" "False"]
  ```
What are these $?\text{-variables}$?

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

Example: theorem $\text{conjI}: [?P; ?Q] \Longrightarrow ?P \land ?Q$

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

- By hand:
  
  $\text{conjI[of "a=b" "False"]} \leadsto
  \begin{array}{l}
  \llbracket a = b; False \rrbracket \Longrightarrow a = b \land False
  \end{array}$

- By unification:
  
  unifying $?P \land ?Q$ with $a = b \land False$

Rule application

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

subgoal: 1. $\ldots \Longrightarrow A \land B$
Rule application

Example: rule: \[ ?P; ?Q \] \implies ?P \land ?Q 
subgoal: 1. \ldots \implies A \land B 
Result: 1. \ldots \implies A 
2. \ldots \implies B 

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

Rule application

Example: rule: \[ ?P; ?Q \] \implies ?P \land ?Q 
subgoal: 1. \ldots \implies A \land B 
Result: 1. \ldots \implies A 
2. \ldots \implies B 

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

apply(rule xyz)
Rule application

Example: rule: \[ [P, Q] \implies P \land Q \]
subgoal: 1. \[ \ldots \implies A \land B \]
Result: 1. \[ \ldots \implies A \]
2. \[ \ldots \implies B \]

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

\textbf{apply}(\textit{rule xyz})
"Backchaining"

Typical backwards rules

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

\[ \frac{P \implies Q}{P \implies P \land Q} \text{\textit{impI}} \]

Typical backwards rules

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

\[ \frac{P \implies Q}{P \implies P \land Q} \text{\textit{impI}} \]

\[ \frac{\forall x. P x}{\exists x. P x} \text{\textit{allI}} \]
Typical backwards rules

\[
\frac{\neg P \quad \neg Q}{\neg P \land \neg Q} \text{ conjI} \\
\frac{\neg P \rightarrow \neg Q}{\neg P \rightarrow \neg Q} \text{ impI} \\
\frac{\bigwedge x. \neg P_x}{\neg \forall x. \neg P_x} \text{ allI} \\
\frac{\neg P \rightarrow \neg Q \quad \neg Q \rightarrow \neg P}{\neg P = \neg Q} \text{ iffI}
\]

Forward proof: OF

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

\[
\text{Forward proof: OF}
\]

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

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

is the theorem obtained by proving \( A \) with \( s \).
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: \(?t = ?t\)

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

\[
\sim^\updownarrow
\]

\(?Q \implies a = a \land ?Q\)
The general case:

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

\[
\text{r[OF r}_1 \ldots \text{ r}_m]
\]

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

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

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

\( a = a \land b = b \)

From now on: \( ? \) mostly suppressed on slides
Case distinction

\[
\begin{align*}
\text{show} & \quad R \\
\text{proof} & \quad \text{cases} \\
& \quad \text{assume} \quad P \\
& \quad : \\
& \quad \text{show} \quad R \quad \ldots \\
\text{next} & \quad \text{assume} \quad \neg P \\
& \quad : \\
& \quad \text{show} \quad R \quad \ldots \\
\text{qed} & \\
\text{have} & \quad P \lor Q \quad \ldots \\
\text{then} & \quad \text{show} \quad R \\
\text{proof} & \\
& \quad \text{assume} \quad P \\
& \quad : \\
& \quad \text{show} \quad R \quad \ldots \\
\text{next} & \quad \text{assume} \quad Q \\
& \quad : \\
& \quad \text{show} \quad R \quad \ldots \\
\text{qed}
\end{align*}
\]