

CS576 Topics in Automated Deduction

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu

<http://courses.grainger.illinois.edu/cs576>

Slides based in part on slides by Tobias Nipkow

February 17, 2026

Rewriting with Definitions (definition)

Definitions do not have the `simp` attribute.

They must be used explicitly:

```
proof (simp add: f_def ...)
```

Ordered Rewriting

Problem: $?x+?y=?y+?x$ does not terminate

Solution: Permutative `simp`-rules are used only if the term becomes lexicographically smaller.

Example: $b + a \rightsquigarrow a + b$ but not $a + b \rightsquigarrow b + a$.

For types `nat`, `int`, etc., commutative, associative and distributive laws built in.

Example: `proof simp` yields:

```
((B + A) + ((2 :: nat) * C)) + (A + B) ~  
... ~ 2 * A + (2 * B + 2 * C)
```

Preprocessing

`simp`-rules are preprocessed (recursively) for maximal simplification power:

```
 $\neg A \mapsto A = \text{False}$   
 $A \mapsto B \mapsto A \implies B$   
 $A \wedge B \mapsto A, B$   
 $\forall x. A(x) \mapsto A(?x)$   
 $A \mapsto A = \text{True}$ 
```

Example:

```
(p  $\implies$  q  $\wedge$   $\neg$ r)  $\wedge$  s  $\mapsto$  p  $\implies$  q = True,  
p  $\implies$  r = False,  
s = True
```

Demo: Simplification through Rewriting

General Isar Proof Format

```
proof (method)  
  fix x  
  assume A0: formula0  
  from A0  
  have A1: formula1  
  by (method)  
  from A0 and A1  
  ...  
  show formulan  
  proof (method)  
    ...  
  qed  
qed
```

Proves $formula_0 \implies formula_n$

Basic Isar Syntax

```

proof = proof [method] statement* qed
      | by method

method = (simp ...)|(auto ...)|(blast ...)|(rule ...) | ...

statement = fix variable+ (Λ)
            | assume proposition (⇒)
            | [from name+] objective proof
            | next (starts next subgoal)

objective = show proposition (next proof step)
            | have proposition (local claim)
            | obtain variable+ where proposition+

proposition = [name:] formula
    
```

Proof Basics

- Isabelle uses *Natural Deduction* proofs
 - Uses *sequent* encoding
- Rule notation:

Rule	Sequent Encoding
$\frac{A_1 \dots A_n}{A}$	$\llbracket A_1, \dots, A_n \rrbracket \Rightarrow A$
$\frac{B \quad \vdots \quad A_1 \dots \frac{A_i}{A} \dots A_n}{A}$	$\llbracket A_1, \dots, B \rrbracket \Rightarrow \llbracket A_i, \dots, A_n \rrbracket \Rightarrow A$

Natural Deduction

For each logical operator \oplus , have two kinds of rules:

Introduction: How can I prove $A \oplus B$?

$$\frac{?}{A \oplus B}$$

Elimination: What can I prove using $A \oplus B$?

$$\frac{\dots A \oplus B \dots}{?}$$

Operational Reading

$$\frac{A_1 \dots A_n}{A}$$

Introduction rule:

To prove A it suffices to prove $A_1 \dots A_n$.

Elimination rule:

If we know A_1 and we want to prove A it suffices to prove $A_2 \dots A_n$

Natural Deduction for Propositional Logic

$\frac{A \quad B}{A \wedge B} \text{ conjI}$	$\frac{A \wedge B \quad \llbracket A; B \rrbracket \Rightarrow C}{C} \text{ conjE}$
$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{ disjI1/2}$	$\frac{A \vee B \quad A \Rightarrow C \quad B \Rightarrow C}{C} \text{ disjE}$
$\frac{A \Rightarrow B}{A \rightarrow B} \text{ impI}$	$\frac{A \rightarrow B \quad A \quad B \Rightarrow C}{C} \text{ impE}$

Natural Deduction for Propositional Logic

$\frac{A \Rightarrow B \quad B \Rightarrow A}{A = B} \text{ iffI}$	$\frac{A = B \quad A}{B} \text{ iffD1}$
	$\frac{A = B \quad B}{A} \text{ iffD2}$
$\frac{A \Rightarrow \text{False}}{\neg A} \text{ notI}$	$\frac{\neg A \quad A}{B} \text{ notE}$

Equality

$$\frac{}{t = t} \text{ refl} \quad \frac{s = t}{t = s} \text{ sym} \quad \frac{r = s \quad s = t}{r = t} \text{ trans}$$

$$\frac{s = t \quad A(s)}{A(t)} \text{ subst}$$

`subst` rarely needed explicitly – used implicitly by `simp`

More Rules

$$\frac{A \wedge B}{A} \text{ conjunct1} \quad \frac{A \wedge B}{B} \text{ conjunct2}$$

$$\frac{A \rightarrow B \quad A}{B} \text{ mp}$$

Compare to elimination rules:

$$\frac{A \wedge B \quad [A; B] \Rightarrow C}{C} \text{ conjE} \quad \frac{A \rightarrow B \quad A \quad B \Rightarrow C}{C} \text{ impE}$$

“Classical” Rules

$$\frac{\neg A \Rightarrow \text{False}}{A} \text{ ccontr} \quad \frac{\neg A \Rightarrow A}{A} \text{ classical}$$

- `ccontr` and `classical` are not derivable from the Natural Deduction rules.
- They make the logic “classical”, i.e. “non-constructive or non-intuitionistic”.

Proof by Assumption

In classical Natural Deduction,

$$\frac{A_1 \dots A_i \dots A_n}{A_i}$$

If we know a bunch of things, including A_i , then we know A_i

In Isabelle

$$[A] \Rightarrow A$$

Rule Application: The Rough Idea

Applying rule $[A_1; \dots; A_n] \Rightarrow A$ to subgoal C :

- Unify A and C
- Replace C with n new subgoals: $A'_1 \dots A'_n$

Backwards reduction, like in Prolog

Example: rule: $[?P; ?Q] \Rightarrow ?P \wedge ?Q$

subgoal: 1. $A \wedge B$

Result: 1. A
2. B

Rule Application: More Complete Idea

Applying rule $[A_1; \dots; A_n] \Rightarrow A$ to subgoal C :

- Unify A and C with (meta)-substitution σ
- Specialize goal to $\sigma(C)$
- Replace C with n new subgoals: $\sigma(A_1) \dots \sigma(A_n)$

Note: schematic variables in C treated as existential variables

Does there exist value for $?X$ in C that makes C true?

(Still not the whole story)

rule Application

Rule: $\llbracket A_1; \dots; A_n \rrbracket \Longrightarrow A$

Subgoal: 1. $\llbracket B_1; \dots; B_m \rrbracket \Longrightarrow C$

Substitution: $\sigma(A) \equiv \sigma(C)$

New subgoals: 1. $\llbracket \sigma(B_1); \dots; \sigma(B_m) \rrbracket \Longrightarrow \sigma(A_1)$
 \vdots
 n. $\llbracket \sigma(B_1); \dots; \sigma(B_m) \rrbracket \Longrightarrow \sigma(A_n)$

Proves: $\llbracket \sigma(B_1); \dots; \sigma(B_m) \rrbracket \Longrightarrow \sigma(C)$

Command: `apply (rule <rule-name>)`

Proof by assumption

apply assumption

proves:

1. $\llbracket B_1; \dots; B_m \rrbracket \Longrightarrow C$

by unifying C with one of the B_i

Demo: Application of Introduction Rule

Applying Elimination Rules

apply (erule <elim-rule>)

Like `rule` but also

- unifies first premise of rule with an assumption
- eliminates that assumption instead of conclusion
- `proof (rule ...)` generally does the work of `erule` in Isar.

Example

Rule: $\llbracket ?P \wedge ?Q; \llbracket ?P; ?Q \rrbracket \Longrightarrow ?R \rrbracket \Longrightarrow ?R$

Subgoal: 1. $\llbracket X; A \wedge B; Y \rrbracket \Longrightarrow Z$

Unification: $?P \wedge ?Q \equiv A \wedge B$ and $?R \equiv Z$

New subgoal: 1. $\llbracket X; Y \rrbracket \Longrightarrow \llbracket A; B \rrbracket \Longrightarrow Z$

Same as: 1. $\llbracket X; Y; A; B \rrbracket \Longrightarrow Z$

How to Prove in Natural Deduction

- *Intro* rules decompose formulae to the *right* of \Longrightarrow
`apply (rule <intro-rule>)`
`proof (rule <intro-rule>)`
- *Elim* rules decompose formulae to the *left* of \Longrightarrow
`apply (erule <elim-rule>)`
`proof (rule <elim-rule>)`

Demo: Examples

Safe and Unsafe Rules - Basic Logical Connectives

Safe rules preserve provability:

`conjI, impI, notI, iffI, refl, ccontr, classical, conjE, disjE`

Unsafe rules can reduce a provable goal to one that is not:

`disjI1, disjI2, impE, iffD1, iffD2, notE`

Try safe rules before unsafe ones

\implies VS \rightarrow

- Theorems usually more useful written as $[A_1; \dots; A_n] \implies A$ instead of $A_1 \wedge \dots \wedge A_n \rightarrow A$ (easier to apply)
- **Exception:** (in `apply`-style): induction variable must not occur in premises
- Example: For induction on x , transform:
 $[A; B(x)] \implies C(x) \rightsquigarrow A \implies B(x) \rightarrow C(x)$
Reverse transformation (after proof):
`lemma abc [rule_format]: A \implies B(x) \rightarrow C(x)`

Demo: Further Techniques

Parameters

Subgoal:

1. $\Lambda x_1 \dots x_n. \text{Formula}$

The x_i are called *parameters* of the subgoal
Intuition: local constants, i.e. arbitrary fixed values

Rules are automatically lifted passed $\Lambda x_1 \dots x_n$ and applied directly to *Formula*

Scope

- Scope of parameters: whole subgoal
- Scope of \forall, \exists, \dots : ends with $;$ or \implies , or enclosing $)$

$\Lambda xy. [\forall y. P y \rightarrow Q z y; Q x y] \implies \exists x. Q x y$
means
 $\Lambda xy. [(\forall y_1. P y_1 \rightarrow Q z y_1); Q x y] \implies \exists x_1. Q x_1 y$

α -Conversion and Scope of Variables

- $\forall x. P x$: x can appear in $P x$.
Example: $\forall x. x = x$ is obtained by $P \mapsto \lambda u. u = u$
- $\forall x. P$: x cannot appear in P
Example: $P \mapsto x = x$ yields $\forall x'. x = x$

Bound variables are renamed automatically to avoid name clashes with other variables.

Natural Deduction Rules for Quantifiers

$$\frac{\Lambda x. P x}{\forall x. P x} \text{allI} \qquad \frac{\forall x. P x \quad P ?x \implies R}{R} \text{allE}$$
$$\frac{P ?x}{\exists x. P x} \text{exI} \qquad \frac{\exists x. P x \quad \Lambda x. P x \implies R}{R} \text{exE}$$

- **allI** and **exE** introduce new parameters (Λx)
- **allE** and **exI** introduce new unknowns ($?x$)

Safe and Unsafe Rules - Quantifiers

Safe: **allI**, **exE**

Unsafe: **allE**, **exI**

Create parameters first, unknowns later

Instantiating Variables in Rules

```
proof (rule_tac x = "term" in rule)
```

Like **rule**, but $?x$ in *rule* is instantiated with *term* before application.
 $?x$ must be schematic variable occurring in statement of *rule*.

Similar: **erule_tac**

! x is in *rule*, not in goal !