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

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

$\Longrightarrow$ vs $\longrightarrow$

- Theorems usually more useful written as

$$[\![A_1; \ldots; A_n]\!] \Longrightarrow A$$

  instead of $A_1 \wedge \ldots \wedge A_n \longrightarrow A$ (easier to apply)

- Exception: (in `apply`-style): induction variable must not occur in premises
- Example: For induction on x, transform:

$$[\![A; B(x)]\!] \Longrightarrow C(x) \quad \rightsquigarrow \quad A \Longrightarrow B(x) \longrightarrow C(x)$$

- Reverse transformation (after proof):

```
lemma abc [rule_format]: A ⟹ B(x) ⟶ C(x)
```

Demo: Further Techniques

α-Conversion and Scope of Variables

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

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

Natural Deduction Rules for Quantifiers

$$\frac{\Lambda x.\ P\ x}{\forall x.P\ x}\ \texttt{allI} \qquad \frac{\forall x.\ P\ x \quad P\ ?x \Longrightarrow R}{R}\ \texttt{allE}$$

$$\frac{P\ ?x}{\exists x.\ P\ x}\ \texttt{exI} \qquad \frac{\exists x.\ P\ x \quad \Lambda x.\ P\ x \Longrightarrow R}{R}\ \texttt{exE}$$

- 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 !

Three Apply-Style Successful Proofs

```
1. ∀x. ∃y. x = y
apply (rule allI)
1. ⋀x.∃y. x = y
```

Better practice:

```
apply(rule_tac x = "x" in exI)
1. ⋀x. x = x
apply (rule refl)
```

simpler & cleaner

Exploration:

```
apply (rule exI)
1. ⋀x. x =?yx
apply (rule refl)
?y ↦ λu. u
```

shorter & trickier

Successful Attempt in Isar

```
lemma shows "∀(x::'a). ∃y. x = y"
proof (rule allI)
 fix x::'a
 show "∃y. x = y"
 proof (rule exI)
  show "x = x" by (rule refl)
 qed
qed
```

Two Unsuccessful Apply-Style Proof Attempts

```
                              1. ∃y. ∀x. x = y
apply(rule_tac                    apply (rule exI)
      x = ??? in exI)             1. ∀x. x =?y
                                  apply(rule allI)
                                  1. Λx. x =?y
                                  apply(rule refl)
                                  ?y ↦ x yields Λx'. x' = x
```

Principles: $?f\ x_1 \ldots x_n$ can only be replaced by term t if $params(t) \subseteq \{x_1, \ldots, x_n\}$

Parameter Names

Parameter names are chosen by Isabelle

```
1. ∀x. ∃y. x = y
apply(rule allI)
1. ⋀x. ∃y. x = y
apply(rule_tac x = "x" in exI)
```

Works, but is brittle!!

Better to use Isar, where you choose the name.

Forward Proofs: `frule` and `drule`

"Forward" rule:	$\mathtt{A_1} \Longrightarrow \mathtt{A}$
Subgoal:	$1.\ [\![\mathtt{B_1}; \ldots; \mathtt{B_n}]\!] \Longrightarrow \mathtt{C}$
Substitution:	$\sigma(\mathtt{B}_i) \equiv \sigma(\mathtt{A_1})$
New subgoal:	$1.\ \sigma([\![\mathtt{B_1}; \ldots; \mathtt{B_n}; \mathtt{A}]\!] \Longrightarrow \mathtt{C})$

Command:

`apply(frule` $<$ *rulename* $>$`)`

Like `frule` but also deletes $\mathtt{B}_i$:

`apply(drule` $<$ *rulename* $>$`)`

frule and drule: The General Case

Rule: $[\![\mathtt{A}_1; \ldots; \mathtt{A}_m]\!] \Longrightarrow \mathtt{A}$
Creates additional subgoals:

$$
\begin{array}{l}
1.\ \sigma([\![\mathtt{B}_1; \ldots; \mathtt{B}_n]\!] \Longrightarrow \mathtt{A}_2) \\
\quad \vdots \\
m-1.\ \sigma([\![\mathtt{B}_1; \ldots; \mathtt{B}_n]\!] \Longrightarrow \mathtt{A}_m) \\
m.\ \sigma([\![\mathtt{B}_1; \ldots; \mathtt{B}_n; \mathtt{A}]\!] \Longrightarrow \mathtt{C})
\end{array}
$$

Forward Proofs: OF

$$\texttt{r [OF } \texttt{r}_1 \ldots \texttt{r}_n\texttt{]}$$

Prove assumption 1 of theorem r with theorem $\texttt{r}_1$, and assumption 2 with theorem $\texttt{r}_2$, etc.

Rule r	$[\![\texttt{A}_1; \ldots; \texttt{A}_\texttt{m}]\!] \Longrightarrow \texttt{A}$
Rule $\texttt{r}_1$	$[\![\texttt{B}_1; \ldots; \texttt{B}_\texttt{n}]\!] \Longrightarrow \texttt{B}$
Substitution	$\sigma(\texttt{B}) \equiv \sigma(\texttt{A}_1)$
r [OF $\texttt{r}_1$]	$\sigma([\![\texttt{B}_1; \ldots; \texttt{B}_\texttt{n}; \texttt{A}_2; \ldots; \texttt{A}_\texttt{m}]\!] \Longrightarrow \texttt{A})$

Forwards Proofs: THEN

r_1[THEN r_2] means r_2[OF r_1]

Forward Proofs: of

Given a theorem like gcd_mult_distrib2:

```
?k * gcd (?m, ?n) = gcd (?k * ?m, ?k * ?n)
```

We want to replace ?m by 1.

of instantiates variables left to right

In above the order is ?k, ?m, and ?n

[of k 1] replaces ?k by k, and ?m by 1.

gcd_mult_distrib2 [of k 1] yields

```
k * gcd (1, ?n) = gcd (k * 1, k * ?n)
```

Forward Proofs: `where`

Alternately, with `where` you can specify the variable to get the term:

`gcd_mult_distrib2 [where m = "1"]` yields

```
?k * gcd (1, ?n) = gcd (?k * 1, ?k * ?n)
```

Same result given by `gcd_mult_distrib2 [of _ 1]`

and `gcd_mult_distrib2 [where m = "1" and k = "k"]` yields

```
k * gcd (1, ?n) = gcd (k * 1, k * ?n)
```

Caution: `of` and `where` cannot use goal parameters

Forward Proofs: `lemmas`

- Can use `lemmas` to capture result of forward proof:
  `lemmas gcd_mult0 = gcd_mult_distrib2 [of k 1]`
- Can follow on with more forward reasoning:
  `lemmas gcd_mult1 = gcd_mult0 [simplified]` yields
  `k = gcd (k, k * ?n)`
- `[simplified]` applies `simp` to theorem

Forward Proofs: `lemmas`

- Can combine multiple steps together:
  ```
  lemmas gcd_mult =
  gcd_mult_distrib2 [of _ 1, simplified, THEN sym]
  ```
  yields
  ```
  gcd (?k, ?k * ?n) = ?k
  ```

Adding Assumptions to Goals

- `insert` *thm* insert *thm* as new assumption to current subgoal

```
lemma relprime_dvd_mult:
"⟦gcd(k,n) = 1;  k dvd m * n⟧ ⟹ k dvd m"
apply (insert gcd_mult_distrib2 [of m k n])
```

yields:

```
⟦gcd(k,n) = 1;  k dvd m * n;  m * gcd(k,n) = gcd(m * k,m * n)⟧ ⟹ k dvd
```

Adding Assumptions to Goals

Note: `of` and `where` can use only original user variables, but not Isabelle generated parameters

`cut_tac k="m" and m="k" and n="n" in gcd_mult_distrib2` yields same result as above

`cut_tac` can use parameters

Adding Assumptions to Goals: `subgoal_tac`

- Can always add assumption *asm* to current subgoal with `apply (subgoal_tac "asm")`
- Statement can use Isabelle parameters
- Adds new subgoal *asm* with same assumptions as current subgoal

Adding Assumptions to Goals: subgoal_tac

1. $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$

```
apply (subgoal_tac "asm")
```

yields

1. $[\![A_1; \ldots; A_n; asm]\!] \Longrightarrow A$
2. $[\![A_1; \ldots; A_n]\!] \Longrightarrow asm$

Removing Assumptions: thin_tac

- Can remove unwanted assumption *asm* from current subgoal with `apply (thin_tac "asm")`

1. $[\![A_1; \ldots; A_{i-1}; A_i; A_{i+1}; \ldots; A_n]\!] \Longrightarrow A$

`apply (thin_tac "A`$_i$`")`

yields

1. $[\![A_1; \ldots; A_{i-1}; A_{i+1}; \ldots; A_n; \mathtt{asm}]\!] \Longrightarrow A$

"Clarifying" the Goal

- `proof (intro ... )`
  Repeated application of intro rules
  Example: `proof (intro allI)`
- `proof (elim ... )`
  Repeated application of elim rules
  Example: `proof (elim conjE)`
- `proof (clarify)`
  Repeated application of safe rules without splitting goal
- `proof (clarsimp simp add: ... )`
  Combination of `clarify` and `simp`

Other Automated Proof Methods

- `blast` Isabelle's most powerful classical reasoner.
  Useful for goals stated using only predicate logic and set theory
  Can be extended with rules (with `[iff]` attribute) to handler broader classes of goals
- `auto`
  Applies to all subgoals.
  Combines classical reasoning with simplification
  Does what it can; leaves unfinished subgoals
  Splits subgoals
- `force`
  Similar to `auto`, but only applies to one goal, and either finishes or fails.
- `safe`
  Like `clarify` but also splits goals

Demo: Proof Methods