

## 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  $\Rightarrow$   
`apply (rule <intro-rule>)`  
`proof (rule <intro-rule>)`
- *Elim* rules decompose formulae to the *left* of  $\Rightarrow$   
`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

## $\Rightarrow$ VS $\rightarrow$

- Theorems usually more useful written as

$$\llbracket A_1; \dots; A_n \rrbracket \Rightarrow A$$

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:

$$\llbracket A; B(x) \rrbracket \Rightarrow C(x) \rightsquigarrow A \Rightarrow B(x) \rightarrow C(x)$$

- Reverse transformation (after proof):

`lemma abc [rule_format]: A  $\Rightarrow$  B(x)  $\rightarrow$  C(x)`

## Demo: Further Techniques

## $\alpha$ -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 ( $\Lambda x$ )
- **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.  $\forall x. \exists y. x = y$   
`apply (rule allI)`  
1.  $\Lambda x. \exists y. x = y$

Better practice:  
`apply(rule_tac x = "x" in exI)`

1.  $\Lambda x. x = x$   
`apply (rule refl)`

simpler & cleaner

Exploration:  
`apply (rule exI)`  
1.  $\Lambda x. x = ?yx$   
`apply (rule refl)`  
 $?y \mapsto \lambda u. u$

shorter & trickier

## Successful Attempt in Isar

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

## Two Unsuccessful Apply-Style Proof Attempts

```

1.  $\exists y. \forall x. x = y$ 
apply(rule_tac
  x = ??? in exI)
  apply (rule exI)
  1.  $\forall x. x = ?y$ 
  apply(rule allI)
  1.  $\Lambda x. x = ?y$ 
  apply(rule refl)
  ?y  $\mapsto$  x yields  $\Lambda x'. x' = x$ 

```

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

## Parameter Names

Parameter names are chosen by Isabelle

```

1.  $\forall x. \exists y. x = y$ 
apply(rule allI)
1.  $\Lambda x. \exists y. x = y$ 
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:  $A_1 \implies A$   
 Subgoal: 1.  $\llbracket B_1; \dots; B_n \rrbracket \implies C$   
 Substitution:  $\sigma(B_j) \equiv \sigma(A_1)$   
 New subgoal: 1.  $\sigma(\llbracket B_1; \dots; B_n; A \rrbracket \implies C)$

Command:

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

Like `frule` but also deletes  $B_j$ :

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

## frule and drule: The General Case

Rule:  $\llbracket A_1; \dots; A_m \rrbracket \implies A$   
 Creates additional subgoals:

```

1.  $\sigma(\llbracket B_1; \dots; B_n \rrbracket \implies A_2)$ 
...
m-1.  $\sigma(\llbracket B_1; \dots; B_n \rrbracket \implies A_m)$ 
m.  $\sigma(\llbracket B_1; \dots; B_n; A \rrbracket \implies C)$ 

```

## Forward Proofs: OF

```
r [OF r1...rn]
```

Prove assumption 1 of theorem  $r$  with theorem  $r_1$ , and assumption 2 with theorem  $r_2$ , etc.

Rule  $r$   $\llbracket A_1; \dots; A_m \rrbracket \implies A$   
 Rule  $r_1$   $\llbracket B_1; \dots; B_n \rrbracket \implies B$   
 Substitution  $\sigma(B) \equiv \sigma(A_1)$   
 $r$  [OF  $r_1$ ]  $\sigma(\llbracket B_1; \dots; B_n; A_2; \dots; A_m \rrbracket \implies A)$

## Forwards Proofs: THEN

$r_1$  [THEN  $r_2$ ] means  $r_2$  [OF  $r_1$ ]

## Forward Proofs: of

Given a theorem like `gcd_mult_distrib2`:

$$?k * \text{gcd} (?m, ?n) = \text{gcd} (?k * ?m, ?k * ?n)$$

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  
 $k * \text{gcd} (1, ?n) = \text{gcd} (k * 1, k * ?n)$

## Forward Proofs: where

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

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

$$?k * \text{gcd} (1, ?n) = \text{gcd} (?k * 1, ?k * ?n)$$

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

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

$$k * \text{gcd} (1, ?n) = \text{gcd} (k * 1, k * ?n)$$

**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  
 $k = \text{gcd} (k, k * ?n)$
- `[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  
 $\text{gcd} (?k, ?k * ?n) = ?k$

## Adding Assumptions to Goals

• `insert_thm` `insert_thru` as new assumption to current subgoal  
`lemma prime_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.  $[A_1; \dots; A_n] \implies A$   
`apply (subgoal_tac "asm")`

yields

1.  $[A_1; \dots; A_n; \text{asm}] \implies A$   
2.  $[A_1; \dots; A_n] \implies \text{asm}$

## Removing Assumptions: `thin_tac`

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

1.  $[A_1; \dots; A_{i-1}; A_i; A_{i+1}; \dots; A_n] \implies A$   
`apply (thin_tac "Ai")`

yields

1.  $[A_1; \dots; A_{i-1}; A_{i+1}; \dots; A_n; \text{asm}] \implies A$

## "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 handle 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