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 19, 2026

Forward Proofs: `frule` and `drule`

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

Command:

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

Like `frule` but also deletes B_i:

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

`frule` and `drule`: The General Case

Rule: $[\![A_1; \ldots; A_m]\!] \Longrightarrow A$

Creates additional subgoals:

$1.\ \sigma([\![B_1; \ldots; B_n]\!] \Longrightarrow A_2)$
$\vdots$
$m-1.\ \sigma([\![B_1; \ldots; B_n]\!] \Longrightarrow A_m)$
$m.\ \sigma([\![B_1; \ldots; B_n; A]\!] \Longrightarrow C)$

In Isar style, use `have`

Forward Proofs: `OF` and `THEN`

`r [OF` $r_1 \ldots r_n$`]`

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

Rule `r`	$[\![A_1; \ldots; A_m]\!] \Longrightarrow A$
Rule r_1	$[\![B_1; \ldots; B_n]\!] \Longrightarrow B$
Substitution	$\sigma(B) \equiv \sigma(A_1)$
`r [OF` r_1`]`	$\sigma([\![B_1; \ldots; B_n; A_2; \ldots; A_m]\!] \Longrightarrow A)$

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]`

`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

- cut_tac *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 (cut_tac 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 m

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; ...; A_n⟧ ⟹ A

apply (subgoal_tac "*asm*")

yields

1. ⟦A_1; ...; A_n; asm⟧ ⟹ A
2. ⟦A_1; ...; A_n⟧ ⟹ 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