

Topics in Automated Deduction (CS 576)

Elsa L. Gunter

2112 Siebel Center

egunter@illinois.edu

<http://www.cs.illinois.edu/class/sp10/cs576/>

Natural Deduction Rules for Quantifiers

$$\frac{\Lambda x. P x}{\forall x. P x} \text{allI}$$

$$\frac{\forall x. P x \quad P ?x \implies R}{R} \text{allE}$$

$$\frac{P ?x}{\exists x. P x} \text{exI}$$

$$\frac{\exists x. P x \quad \Lambda x. P x \implies R}{R} \text{exE}$$

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

Safe and Unsafe Rules

Safe: `allI`, `exE`

Unsafe: `allE`, `exI`

Create parameters first, unknowns later

Instantiating Variables in Rules

```
apply (rule_tac  $x = \text{"term"}$  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 !

Two 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. $\Lambda x. x = ?y \ x$
`apply (rule refl)`
 $?y \mapsto \lambda u. u$

shorter & trickier

Two Unsuccessful 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. $\wedge x. x = ?y$

`apply(rule refl)`

`?y \mapsto x yields $\wedge x'. x' = x$`

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!!

Renaming Parameters

1. $\forall x. \exists y. x = y$

`apply(rule allI)`

1. $\Lambda x. \exists y. x = y$

`apply(rename_tac xxx)`

1. $\Lambda xxx. \exists y. xxx = y$

`apply(rule_tac x = "xxx" in exI)`

In general: `(rename_tac $x_1 \dots x_n$)` renames the right-most (inner) n parameters to $x_1 \dots x_n$

Forward Proofs: `frule` and `drule`

“Forward” rule: $A_1 \implies A$

Subgoal: 1. $\llbracket B_1; \dots; B_n \rrbracket \implies C$

Substitution: $\sigma(B_i) \equiv \sigma(A_1)$

New subgoal: 1. $\sigma(\llbracket B_1; \dots; B_n; A \rrbracket \implies C)$

Command:

`apply(frule < rulename >)`

Like `frule` but also deletes B_i :

`apply(drule < rulename >)`

frule and drule: The General Case

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

Creates additional subgoals:

$$1. \sigma(\llbracket B_1; \dots; B_n \rrbracket \implies A_2)$$

\vdots

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

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

Forward Proofs: OF

$$r \text{ [OF } r_1 \dots r_n \text{]}$$

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

$$\text{Rule } r \quad \llbracket A_1; \dots; A_m \rrbracket \implies A$$

$$\text{Rule } r_1 \quad \llbracket B_1; \dots; B_n \rrbracket \implies B$$

$$\text{Substitution} \quad \sigma(B) \equiv \sigma(A_1)$$

$$r \text{ [OF } r_1 \text{]} \quad \sigma(\llbracket B_1; \dots; B_n; A_2; \dots; A_m \rrbracket \implies A)$$

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 = 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 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. $\llbracket A_1; \dots; A_n \rrbracket \implies A$
`apply (subgoal_tac "asm")`

yields

1. $\llbracket A_1; \dots; A_n; \text{asm} \rrbracket \implies A$
2. $\llbracket A_1; \dots; A_n \rrbracket \implies \text{asm}$

Removing Assumptions: `thin_tac`

- Can remove unwanted assumption *asm* from current subgoal with

```
apply (thin_tac "asm")
```

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

yields

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

“Clarifying” the Goal

- `apply (intro ...)`

Repeated application of intro rules

Example: `apply (intro allI)`

- `apply (elim ...)`

Repeated application of elim rules

Example: `apply (elim conjE)`

- `apply (clarify)`

Repeated application of safe rules without splitting goal

- `apply (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

Sets

Type 'a set: sets over type 'a

- $\{ \}, \{e_1, \dots, e_n\}, \{x. Px\}$
- $e \in A, A \subseteq B$
- $A \cup B, A \cap B, A - B, -A$
- $\bigcup_{x \in A} B \ x, \bigcap_{x \in A} B \ x$
- $\{i. .j\}$
- $\text{insert} :: 'a \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$

- $f \text{ ' } A \equiv \{y. \exists x \in A. y = f x\}$

-

Proofs about Sets

Natural deduction proof rules:

- **equalityI**: $\llbracket A \subseteq B; B \subseteq A \rrbracket \Longrightarrow A = B$
- **equalityE**: *is a rule* $A = B; \llbracket A \subseteq B; B \subseteq A \rrbracket \Longrightarrow PP$
- **subsetI**: $(\wedge x. x \in A \Longrightarrow x \in B) \Longrightarrow A \subseteq B$
- **subsetD**: $\llbracket A \subseteq B; c \in A \rrbracket \Longrightarrow c \in B$
- **IntI**: $\llbracket c \in A; c \in B \rrbracket \Longrightarrow c \in A \cap B$
- **IntD1**: $c \in A \cap B \Longrightarrow c \in A$

- **IntD2**: $c \in A \cap B \implies c \in B$
- **set_ext**: $(\wedge x. (x \in A) = (x \in B)) \implies A = B$
- ... (see Manual)