

## Topics in Automated Deduction (CS 576)

---

Elsa L. Gunter  
2112 Siebel Center  
egunter@illinois.edu  
<http://www.cs.illinois.edu/class/sp10/cs576/>

1

## More Rules

---

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

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

Compare to elimination rules:

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

2

## "Classical" Rules

---

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

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

3

## Proof by Assumption

---

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

4

## Rule Application: The Rough Idea

---

Applying rule  $[A_1; \dots; A_n] \Longrightarrow A$  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:  $[?P; ?Q] \Longrightarrow ?P \wedge ?Q$   
subgoal: 1.  $A \wedge B$

Result: 1.  $A$   
2.  $B$

5

## Rule Application: More Complete Idea

---

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

- Unify  $A$  and  $C$  with (meta)-substitution  $\sigma$
- 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)

6

### rule Application

---

Rule:  $\llbracket A_1; \dots; A_n \rrbracket \implies A$   
Subgoal: 1.  $\llbracket B_1; \dots; B_m \rrbracket \implies C$   
Substitution:  $\sigma(A) \equiv \sigma(C)$   
New subgoals: 1.  $\llbracket \sigma(B_1); \dots; \sigma(B_m) \rrbracket \implies \sigma(A_1)$   
                  :  
                   $n. \llbracket \sigma(B_1); \dots; \sigma(B_m) \rrbracket \implies \sigma(A_n)$   
Proves:  $\llbracket \sigma(B_1); \dots; \sigma(B_m) \rrbracket \implies \sigma(C)$   
Command: `apply (rule <rule-name>)`

7

### Proof by assumption

---

`apply assumption`

proves:

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

by unifying  $C$  with one of the  $B_i$

8

---

Demo: Application of Introduction Rule

9

### 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

10

### Example

---

Rule:  $\llbracket ?P \wedge ?Q; \llbracket ?P; ?Q \rrbracket \implies ?R \rrbracket \implies ?R$   
Subgoal: 1.  $\llbracket X; A \wedge B; Y \rrbracket \implies Z$   
Unification:  $?P \wedge ?Q \equiv A \wedge B$  and  $?R \equiv Z$   
New subgoal: 1.  $\llbracket X; Y \rrbracket \implies \llbracket A; B \rrbracket \implies Z$   
Same as: 1.  $\llbracket X; Y; A; B \rrbracket \implies Z$

11

### How to Prove in Natural Deduction

---

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

12

---

Demo: Examples

13

## Safe and Unsafe Rules

---

**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**

14

$\Longrightarrow$  vs  $\longrightarrow$

---

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

15

---

Demo: Further Techniques

16

## Parameters

---

Subgoal:

1.  $\Lambda x_1 \dots x_n. Formula$

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*

17

## Scope

---

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

$\Lambda xy. \llbracket \forall y. P y \longrightarrow Q z y; Q x y \rrbracket \Longrightarrow \exists x. Q x y$   
means

$\Lambda xy. \llbracket (\forall y_1. P y_1 \longrightarrow Q z y_1); Q x y \rrbracket \Longrightarrow \exists x_1. Q x_1 y$

18

## $\alpha$ -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.

19

## 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 ( $?x$ )

20

## Safe and Unsafe Rules

---

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

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

**Create parameters first, unknowns later**

21

## Instantiating Variables in Rules

---

`apply (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 !**

22

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

shorter & trickier

23