Topics in Automated Deduction (CS 576)

Elsa L. Gunter

2112 Siebel Center

egunter@illinois.edu

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

More Rules

$$\frac{A \wedge B}{A}\ \texttt{conjunct1} \qquad \frac{A \wedge B}{B}\ \texttt{conjunct2}$$

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

Compare to elimination rules:

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

"Classical" Rules

$$\frac{\neg A \Longrightarrow \mathtt{False}}{A}\ \texttt{ccontr} \qquad \frac{\neg A \Longrightarrow A}{A}\ \texttt{classical}$$

- `ccontr` and `classical` are not derivable from the Natural Deduction rules.

- They make the logic *"classical"*, i.e. *"non-constructive* or *"non-intuitionistic"*.

Proof by Assumption

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

Rule Application: The Rough Idea

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

- Unify A and C
- Replace C with n new subgoals: $A'_1 \ \ldots \ A'_n$

Backwards reduction, like in Prolog

Example: rule: $[\![?\mathtt{P}; ?\mathtt{Q}]\!] \Longrightarrow ?\mathtt{P} \wedge ?\mathtt{Q}$
subgoal: `1. A` $\wedge$ `B`

Result:
```
1.  A
2.  B
```

Rule Application: More Complete Idea

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

- Unify A and C with (meta)-substitution σ
- Specialize goal to $\sigma(C)$
- Replace C with n new subgoals: $\sigma(A_1)$ ... $\sigma(A_n)$

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)

`rule` Application

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

Subgoal: 1. $[\![B_1; \ldots; B_m]\!] \Longrightarrow C$

Substitution: $\sigma(A) \equiv \sigma(C)$

New subgoals: 1. $[\![\sigma(B_1); \ldots; \sigma(B_m)]\!] \Longrightarrow \sigma(A_1)$

$\vdots$

n. $[\![\sigma(B_1); \ldots; \sigma(B_m)]\!] \Longrightarrow \sigma(A_n)$

Proves: $[\![\sigma(B_1); \ldots; \sigma(B_m)]\!] \Longrightarrow \sigma(C)$

Command: `apply (rule` <*rulename*>`)`

Proof by `assumption`

`apply assumption`

proves:

1. $[\![B_1; \ldots; B_m]\!] \Longrightarrow C$

by unifying C with one of the B_i

Demo: Application of Introduction Rule

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

Example

Rule:	$[\![?\mathtt{P} \wedge ?\mathtt{Q}; [\![?\mathtt{P}; ?\mathtt{Q}]\!] \Longrightarrow ?\mathtt{R}]\!] \Longrightarrow ?\mathtt{R}$
Subgoal:	1. $[\![\mathtt{X}; \mathtt{A} \wedge \mathtt{B}; \mathtt{Y}]\!] \Longrightarrow \mathtt{Z}$
Unification:	$?P \wedge ?Q \equiv A \wedge B$ and $?R \equiv Z$
New subgoal:	1. $[\![\mathtt{X}; \mathtt{Y}]\!] \Longrightarrow [\![\mathtt{A}; \mathtt{B}]\!] \Longrightarrow \mathtt{Z}$
Same as:	1.$[\![\mathtt{X}; \mathtt{Y}; \mathtt{A}; \mathtt{B}]\!] \Longrightarrow \mathtt{Z}$

How to Prove in Natural Deduction

- *Intro* rules decompose formulae to the *right* of $\Longrightarrow$

  `apply (rule` <*intro-rule*>`)`

- *Elim* rules decompose formulae to the *left* of $\Longrightarrow$

  `apply (erule` <*elim-rule*>`)`

Demo: Examples

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

$\Longrightarrow$ 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:

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

  Reverse transformation (after proof):

  `lemma abc [rule_format]:` $A \Longrightarrow B(x) \longrightarrow C(x)$

Demo: Further Techniques

Parameters

Subgoal:

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

The x_i are called *parameters* of the subgoal

Intuition: local constants, i.e. arbitrary fixed values

Rules are automatically lifted passed $\Lambda x_1 \ldots x_n$ and applied directly to $Formula$

Scope

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

$$\bigwedge xy.\ [\![\forall y.\ P\ y\ \longrightarrow Q\ z\ y;\ Q\ x\ y]\!] \Longrightarrow \exists x.\ Q\ x\ y$$

means

$$\bigwedge xy.\ [\![(\forall y_1.\ P\ y_1\ \longrightarrow Q\ z\ y_1);\ Q\ x\ y]\!] \Longrightarrow \exists x_1.\ Q\ x_1\ y$$

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

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

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

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

Safe and Unsafe Rules

Safe: `allI`, `exE`

Unsafe: `allE`, `exI`

Create parameters first, unknowns later

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 !

Two Successful Proofs

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

```
apply (rule allI)
```

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

Better practice:

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

1. $\bigwedge$x. x = x

```
apply (rule refl)
```

simpler & cleaner

Exploration:

```
apply (rule exI)
```

1. $\bigwedge$x. x =?yx

```
apply (rule refl)
```

?y $\mapsto$ λu. u

shorter & trickier