Applying rules and theorems

**apply** (rule *theorem*): use when the conclusion of *theorem* matches the conclusion of the current goal

**apply** (erule *theorem*): use when the conclusion of *theorem* matches the conclusion of the current goal and the first premise of *theorem* matches a premise of the current goal

**apply** (frule *theorem*): use when the first premise of *theorem* matches a premise of the current goal

**apply** (drule *theorem*): like frule except it deletes the matching premise

**back**: useful if erule/drule/frule are choosing the wrong premise

**apply** assumption: when the conclusion of the current goal is also a premise

Automated methods

**apply** auto: applies automated tools to look for solution

**apply** force: like auto, but “do or die” (and only applies to the first goal)

**apply** clarify: like auto, but less aggressive

**apply** simp: simplifies current goal using term rewriting

**apply** (simp add :*theorems*): like the simplifier, but tells the simplifier to use additional theorems as well (useful groups of theorems for calculation are ring_simps and field_simps)

**apply** clarsimp: a combination of clarify and simp

**apply** blast: a powerful first-order prover

**apply** arith: automatically solves linear arithmetic problems

Other methods

**apply** (insert *theorem*): adds *theorem* as an additional premise
apply (subgoal_tac formula): adds formula as an additional premise, and also
as a new goal to be proven later

apply (induct_tac variable): splits into the appropriate cases to do induction
on variable (when variable has a natural notion of induction, for instance, it is
a natural number)

apply (rule_tac v_1 = t_1 and ... and v_n = t_n in theorem): like rule, but
allows the certain variables to be chosen manually (also erule_tac,drule_tac, and
frule_tac are analagous)

apply (case_tac ...): splits on cases

Handling equality

apply (subst theorem): applies a substitution (theorem should be an equality)

apply (subst (asm) theorem): applies a substitution to one of the hypotheses

apply (subst (i...j) theorem): applies a substitution at the positions indicated

apply (erule subst): applies a substitution from the hypotheses (useful in
conjunction with insert).

apply (erule subst): applies a substitution from the hypotheses (in the right-to-left direction of the equality).

Logical rules

Propositional Logic:
notI: (A ⇒ False) ⇒ ¬A
notE: [¬A; A] ⇒ B
conjl: [A; B] ⇒ A ∧ B
conjE: [A ∧ B; A; B] ⇒ C
conjunct1: P ∧ Q ⇒ P
conjunct2: P ∧ Q ⇒ Q
context_conjI: [P; P ⇒ Q] ⇒ P ∧ Q
disjI1: A ⇒ A ∨ B
disjI2: A ⇒ B ∨ A
disjCI: (¬Q ⇒ P) ⇒ P ∨ Q
excluded_middle: ¬P ∨ P
disjE: [A ∨ B; A ⇒ C; B ⇒ C] ⇒ C
implI: (A ⇒ B) ⇒ (A → B)
impE: \[ [A \rightarrow B; A] \Rightarrow C \]
impCE: \[ [P \rightarrow Q; \neg P \Rightarrow R; Q \Rightarrow R] \Rightarrow R \]
mp: \[ [A \rightarrow B; A] \Rightarrow B \]
iffI: \[ [A \Rightarrow B; B] \Rightarrow A = B \]
iffE: \[ [A = B; [A \rightarrow B; B \rightarrow A]] \Rightarrow C \] \Rightarrow C

\textit{classical}: \[ (\neg A \Rightarrow A) \Rightarrow A \]
notnotD: \[ \neg
\neg P \Rightarrow P \]
\textit{de Morgan}\_disj: \[ (\neg (P \lor Q)) = (\neg P \land \neg Q) \]
\textit{de Morgan}\_conj: \[ (\neg (P \land Q)) = (\neg P \lor \neg Q) \]
\textit{disj}\_not1: \[ (\neg P \lor Q) = (P \rightarrow Q) \]
\textit{disj}\_not2: \[ (P \lor \neg Q) = (Q \rightarrow P) \]

\textbf{First Order Logic:}
exI: \[ Pa \Rightarrow \exists x. Px \]
exE: \[ [\exists x. Px; \forall x. P x \Rightarrow C] \Rightarrow C \]
allI: \[ (\forall x. P x) \Rightarrow \forall x. P x \]
spec: \[ \forall x. Px \Rightarrow Px \]
allE: \[ [\forall x. Px; P x \Rightarrow R] \Rightarrow R \]

\textbf{Equality:}
sym: \[ x = y \Rightarrow y = x \]
trans: \[ [x = y; y = z] \Rightarrow x = z \]

\textbf{Emacs/Proof General}

"C" stands for the control key, and "C-key" means holding down the control key together with key.

C-k: delete the rest of the line
C-a: jump to the beginning of the current line
C-e: jump to the end of the current line
C-c C-n: process the next line in Isabelle (the next button)
C-c C-u: push back the processed part of the text by one line (the undo button)
C-c C-return: evaluate up to where the cursor is
C-c C-p: show the current state of a proof (for instance, in place of an error message currently being shown)

\textbf{Other tips}

Use the browser pages to find theorems.
You can derive your own theorems, and use them as rules.

Use the "find theorems" command in Proof General.

Under the Proof General menu, if you choose options/electric-terminator, the next line of the proof is sent to Isabelle automatically whenever you end a line with a semicolon.