General Isar Proof Format

```
proof (method)
    fix x
    assume A0: formula_0
    from A0
    have A1: formula_1
    by (method)
    from A0 and A1
      ...
      show formula_n
      proof (method)
        ...
      qed
qed
```

Proves $formula_0 \Longrightarrow formula_n$

Basic Isar Syntax

proof	=	`proof` [*method*] *statement** `qed`	
	|	`by` *method*	
method	=	`(simp ...)` | `(auto ...)` | `(blast ...)` | `(rule ...)` | ...	
statement	=	`fix` *variable*$^+$	($\bigwedge$)
	|	`assume` *proposition*	($\Longrightarrow$)
	|	[`from` *name*$^+$] *objective proof*	
	|	`next`	(starts next subgoal)
objective	=	`show` *proposition*	(next proof step)
	|	`have` *proposition*	(local claim)
	|	`obtain` *variable*$^+$ `where` *proposition*$^+$	
proposition	=	[*name*:] *formula*	

Proof Basics

- Isabelle uses *Natural Deduction* proofs
  - Uses *sequent* encoding
- Rule notation:

Rule — **Sequent Encoding**

$$\frac{A_1 \ldots A_n}{A} \qquad \llbracket A_1, \ldots, A_n \rrbracket \Longrightarrow A$$

$$\frac{A_1 \ldots \quad \begin{array}{c} B \\ \vdots \\ \hline A_i \end{array} \quad \ldots A_n}{A} \qquad \llbracket A_1, \ldots, B \Longrightarrow A_i, \ldots, A_n \rrbracket \Longrightarrow A$$

Natural Deduction

For each logical operator $\oplus$, have two kinds of rules:

Introduction: How can I prove $A \oplus B$?

$$\frac{?}{A \oplus B}$$

Elimination: What can I prove using $A \oplus B$?

$$\frac{\ldots A \oplus B \ldots}{?}$$

Operational Reading

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

Introduction rule:

To prove A it suffices to prove $A_1 \ldots A_n$.

Elimination rule:

If we know A_1 and we want to prove A
it suffices to prove $A_2 \ldots A_n$

Natural Deduction for Propositional Logic

$$\frac{\mathtt{A} \quad \mathtt{B}}{\mathtt{A} \wedge \mathtt{B}}\ \mathtt{conjI} \qquad \frac{\mathtt{A} \wedge \mathtt{B} \quad [\![\mathtt{A}; \mathtt{B}]\!] \Longrightarrow \mathtt{C}}{\mathtt{C}}\ \mathtt{conjE}$$

$$\frac{\mathtt{A}}{\mathtt{A} \vee \mathtt{B}} \qquad \frac{\mathtt{B}}{\mathtt{A} \vee \mathtt{B}}\ \mathtt{disjI1/2} \qquad \frac{\mathtt{A} \vee \mathtt{B} \quad \mathtt{A} \Longrightarrow \mathtt{C} \quad \mathtt{B} \Longrightarrow \mathtt{C}}{\mathtt{c}}\ \mathtt{disjE}$$

$$\frac{\mathtt{A} \Longrightarrow \mathtt{B}}{\mathtt{A} \longrightarrow \mathtt{B}}\ \mathtt{impI} \qquad \frac{\mathtt{A} \longrightarrow \mathtt{B} \quad \mathtt{A} \quad \mathtt{B} \Longrightarrow \mathtt{C}}{\mathtt{C}}\ \mathtt{impE}$$

Natural Deduction for Propositional Logic

$$\frac{\mathtt{A} \Longrightarrow \mathtt{B} \quad \mathtt{B} \Longrightarrow \mathtt{A}}{\mathtt{A} = \mathtt{B}}\ \mathtt{iffI}$$

$$\frac{\mathtt{A} = \mathtt{B} \quad \mathtt{A}}{\mathtt{B}}\ \mathtt{iffD1}$$

$$\frac{\mathtt{A} = \mathtt{B} \quad \mathtt{B}}{\mathtt{A}}\ \mathtt{iffD2}$$

$$\frac{\mathtt{A} \Longrightarrow \mathtt{False}}{\neg \mathtt{A}}\ \mathtt{notI}$$

$$\frac{\neg \mathtt{A} \quad \mathtt{A}}{\mathtt{B}}\ \mathtt{notE}$$

Equality

$$\frac{}{\texttt{t} = \texttt{t}}\ \texttt{refl} \qquad \frac{\texttt{s} = \texttt{t}}{\texttt{t} = \texttt{s}}\ \texttt{sym} \qquad \frac{\texttt{r} = \texttt{s} \quad \texttt{s} = \texttt{t}}{\texttt{r} = \texttt{t}}\ \texttt{trans}$$

$$\frac{\texttt{s} = \texttt{t} \quad \texttt{A(s)}}{\texttt{A(t)}}\ \texttt{subst}$$

`subst` rarely needed explicitly – used implicitly by `simp`

More Rules

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

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

Compare to elimination rules:

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

"Classical" Rules

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

- `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

In classical Natural Deduction,

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

If we know a bunch of things, including A_i, then we know A_i

In Isabelle

$$[\![\mathtt{A}]\!] \Longrightarrow \mathtt{A}$$

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: $[\![\texttt{?P}; \texttt{?Q}]\!] \Longrightarrow \texttt{?P} \wedge \texttt{?Q}$

subgoal: 1. $\texttt{A} \wedge \texttt{B}$

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: $\sigma(A_1)\ \ldots\ \sigma(A_n)$

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

- `proof (rule ...)` generally does the work of `erule` in Isar.

Example

Rule: $[\![?P \wedge ?Q; [\![?P; ?Q]\!] \Longrightarrow ?R]\!] \Longrightarrow ?R$

Subgoal: 1. $[\![X; A \wedge B; Y]\!] \Longrightarrow Z$

Unification: $?P \wedge ?Q \equiv A \wedge B$ and $?R \equiv Z$

New subgoal: 1. $[\![X; Y]\!] \Longrightarrow [\![A; B]\!] \Longrightarrow Z$

Same as: 1. $[\![X; Y; A; B]\!] \Longrightarrow Z$

How to Prove in Natural Deduction

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

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

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

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

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

  `proof (rule` <*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
  $[\![A_1; \dots ; A_n]\!] \Longrightarrow A$
  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:
  $[\![A; B(x)]\!] \Longrightarrow C(x) \leadsto A \Longrightarrow B(x) \longrightarrow C(x)$

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

Demo: Further Techniques

Parameters

Subgoal:

1. $\bigwedge x_1 \dots x_n.$ *Formula*

The x_i are called *parameters* of the subgoal
Intuition: local constants, i.e. arbitrary fixed values

Rules are automatically lifted passed $\bigwedge x_1 \dots x_n$ and applied directly to *Formula*

Scope

- Scope of parameters: whole subgoal
- Scope of $\forall$, $\exists$, $\ldots$: 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

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

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

- `allI` and `exE` introduce new parameters ($\bigwedge x$)
- `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

`proof (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 !