ISAR — a language for structured proofs

Apply scripts versus Isar proofs

unreadable
hard to maintain
do not scale
} NO STRUCTURE!

→ Apply script = assembly language program

→ Isar proof = structured program with comments

But: **apply** still useful for proof exploration

A typical Isar proof

proof
  assume *formula*$_0$
  have *formula*$_1$ **by** *simp*
  ⋮
  have *formula*$_n$ **by** *blast*
  show *formula*$_{n+1}$ **by** ...
qed

] The steps to get from formula$_0$ to formula$_{n+1}$

proves $formula_0 \Longrightarrow formula_{n+1}$

Isar core syntax

proof = **proof** [method] step* **qed**
  | **by** method

method = *(simp ...)* | *(blast ...)* | *(induction ...)* | *(rule ...)* | ...

step = **fix** variables ($\bigwedge$)
  | **assume** prop ($\Longrightarrow$)
  | [**from** fact$^+$] (**have** | **show**) prop proof

use to prove goal

prop = [name:] "formula"

fact = name | ...

Example: Cantor's theorem

Informally: The power set of a set is always larger than the set it originated from.

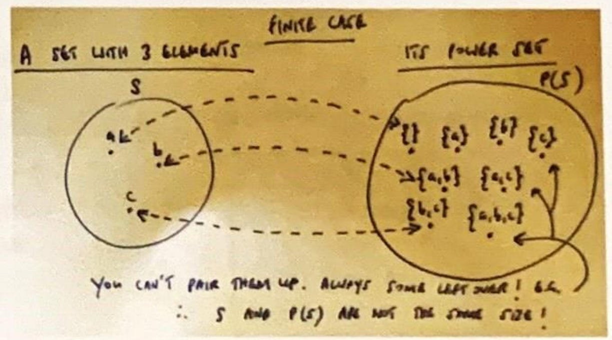

Figure 1: A finite set S and its power-set $\mathcal{P}(S)$. If you can't pair-up elements of sets, with nothing left over, then they cannot be the same size.

SURJECTIVE:
- some elems. will be left in the powerset

Figure from `https://ianwrightsite.files.wordpress.com/2018/10/infinityhandout.pdf`

Example: Cantor's theorem

Informally: The power set of a set is always larger than the set it originated from.

It is not possible to have a fn tht. is surjective from 'a to 'a set

lemma $\neg$ *surj(f :: 'a ⇒ 'a set)*
proof default proof: assume *surj*, show *False*
  assume *a: surj f*
  from *a* **have** *b*: $\forall$ *A*. $\exists$ *a*. *A* = *f a*
    by(*simp add: surj_def*)
  from *b* **have** *c*: $\exists$ *a*. {*x*. *x* $\notin$ *f x*} = *f a*
    by *blast*
  from *c* **show** *False*
    by *blast*
qed

Abbreviations

this	=	the previous proposition proved or assumed
then	=	**from** *this*
thus	=	**then show**
hence	=	**then have**

using and with

(**have**|**show**) prop **using** facts
=
from facts (**have**|**show**) prop

with facts
=
from facts *this*

Structured lemma statement

lemma
  fixes *f* :: "'a ⇒ 'a set"
  assumes *s*: "*surj f*"
  shows "*False*"
proof - no automatic proof step
The use of dash (–)
  have "$\exists$ *a*. {*x*. *x* $\notin$ *f x*} = *f a*" **using** *s*
    by(*auto simp: surj_def*)
  thus "*False*" **by** *blast*
qed

Proves surj f ⟹ False
but surj f becomes local fact s in proof.

The essence of structured proofs

Assumptions and intermediate facts
can be named and referred to explicitly and selectively

Structured lemma statements

fixes $x :: \tau_1$ **and** $y :: \tau_2$ ...
assumes a: P **and** b: Q ...
shows R

- **fixes** and **assumes** sections optional
- **shows** optional if no **fixes** and **assumes**

Proof patterns: Case distinction

excluded-middle

show "R"
proof *cases*
  assume "P"
  ⋮
  show "R" ...
next
  assume "$\neg P$"
  ⋮
  show "R" ...
qed

have "$P \vee Q$" ...
then show "R"
proof
  assume "P"
  ⋮
  show "R" ...
next
  assume "Q"
  ⋮
  show "R" ...
qed

Proof patterns: Contradiction

show "$\neg P$"
proof
  assume "P"
  ⋮
  show "*False*" ...
qed

show "P"
proof (*rule ccontr*)
  assume "$\neg P$"
  ⋮
  show "*False*" ...
qed

Proof patterns: $\longleftrightarrow$

```
show "P ⟷ Q"
proof
  assume "P"
  ⋮
  show "Q" ...
next
  assume "Q"
  ⋮
  show "P" ...
qed
```

Proof patterns: ∀ and ∃ introduction

```
show "∀x. P(x)"
proof
  fix x    local fixed variable
  show "P(x)" ...
qed
```

```
show "∃x. P(x)"
proof
  ⋮
  show "P(witness)" ...
qed
```

Proof patterns: ∃ elimination: obtain

```
have ∃x. P(x)
then obtain x where p: P(x) by blast
⋮   x fixed local variable
```

Works for one or more x

obtain example

```
lemma ¬ surj(f :: 'a ⇒ 'a set)
proof
  assume surj f
  hence ∃a. {x. x ∉ f x} = f a by(auto simp: surj_def)
  then obtain a where {x. x ∉ f x} = f a by blast
  hence a ∉ f a ⟷ a ∈ f a by blast
  thus False by blast
qed
```

Proof patterns: Set equality and subset

show "$A = B$"
proof
  show "$A \subseteq B$" ...
next
  show "$B \subseteq A$" ...
qed

show "$A \subseteq B$"
proof
  fix x
  assume "$x \in A$"
  $\vdots$
  show "$x \in B$" ...
qed

Example: pattern matching

show $formula_1 \longleftrightarrow formula_2$ (**is** $?L \longleftrightarrow ?R$)
proof
  assume $?L$
  $\vdots$
  show $?R$ ...
next
  assume $?R$
  $\vdots$
  show $?L$ ...
qed

?thesis

show *formula* (*is ?thesis*)
proof -
  $\vdots$
  show *?thesis* ...
qed

Every show implicitly defines *?thesis*

let

Introducing local abbreviations in proofs:

let $?t$ = "some-big-term"
$\vdots$
have "...$?t$ ..."

Quoting facts by value

By name:

have *x0*: "$x > 0$" ..
⋮
from *x0* ..

By value:

have "$x > 0$" ..
⋮
from \`x>0\` ..

↑ ↑
back quotes

Example

lemma
"($\exists$ *ys zs. xs = ys @ zs* $\wedge$ *length ys = length zs*) $\vee$
($\exists$ *ys zs. xs = ys @ zs* $\wedge$ *length ys = length zs + 1*)"
proof ???

When automation fails

Split proof up into smaller steps.

Or explore by **apply**:

have .. **using** ..	
apply -	to make incoming facts part of proof state. Note the "-"
apply *auto*	or whatever
apply ..	

At the end:

- **done**
- Better: convert to structured proof

moreover—ultimately

have "P_1" ...
moreover
have "P_2" ...
moreover
⋮
moreover
have "P_n" ...
ultimately
have "P" ...

$\approx$

have lab_1: "P_1" ...
have lab_2: "P_2" ...
⋮
have lab_n: "P_n" ...
from lab_1 lab_2 ...
have "P" ...

With names

Raw proof blocks (Local lemma)

[Prove locally]

$\{$ **fix** $x_1 \dots x_n$
assume $A_1 \dots A_m$
$\vdots$
have B
$\}$

proves $[\![A_1; \dots ; A_m]\!] \Longrightarrow B$
where all x_i have been replaced by $?x_i$.

```
lemma "Q x"
proof -
  fix x
  assume "Q x"
  then show "Q x"
qed
```

NOT allowed

```
lemma "Q x"
proof -
  {fix x
   assume "Q x"
   then have "Q x" by rule}
  then show "Q x"
qed.
```

Summary

- Introduction to Isar and to some common proof patterns e.g. case distinction, contradiction, etc.
- Structured proofs are becoming the norm for Isabelle as they are more readable and easier to maintain.
- Mastering structured proof takes practice and it is usually better to have a clear proof plan beforehand.
- Useful resource: Isar quick reference manual (see AR web page).
- Reading: N&K (Concrete Semantics), Chapter 5.

Proof state and Isar text

In general: **proof** *method*

Applies *method* and generates subgoal(s):

$$\bigwedge x_1 \dots x_n\ [\![A_1; \dots ; A_m]\!] \Longrightarrow B$$

How to prove each subgoal:

fix $x_1 \dots x_n$
assume $A_1 \dots A_m$
$\vdots$
show B

Separated by **next**