The Isar Proof Language in 2016

Makarius Wenzel
August 2016

http://sketis.net

Abstract

This is a description of the Isar proof language as it stands today in 2016. This means the official release Isabelle2016 (February 2016), and the next release that is presumably published towards the end of the year. Relevant NEWS entries and updated portions from the Isabelle/Isar Reference Manual are summarized in one comprehensive article.

1 Introduction

The Isar proof language was originally developed between 1998 and 2001 [1], with the aim to go beyond classic "proof scripts" and support structured *proof texts* or *proof documents*. The original approach turned out sufficiently flexible to last many years without significant reforms. Recently, some loose threads have been picked up again, to address known inconveniences and limitations. Most of this renovated Isar proof language has made it into the release Isabelle2016 (February 2016) already, but some further refinements are for the next release.[1]

The example in figure 1 gives a general impression of Isar in 2016: it is not radically different from what is already known. Note that this formalization of the Schröder-Bernstein Theorem goes back to very early versions of Isabelle, published by Paulson and Nipkow in the 1990s. The cover page of the Isabelle book from 1993 (Springer LNCS 828) depicts the main proof idea. I have rewritten that in the typical Isar style that mixes elementary rule applications and automated reasoning steps.

In the example, structured statements are written in the new notation for explicit *eigen-contexts*: premises with with keyword **if** and parameters with keyword **for**. In the next release, this will work both for toplevel statements (**theorem**, **lemma**) and within the proof (**have**, **show**).

Moreover, inner syntax entities (types, terms, propositions) are uniformly *embedded* into the outer syntax (theories) via *text cartouches* of the form ‹*t*›. This notation has the potential to reduce surprise for new users of Isabelle, or bystanders in Isabelle presentations: the funny cartouche delimiters are hardly ever seen in other languages and thus free from associations about their meaning. Contrast this with traditional

[1] This paper has been processed with Isabelle/ec095a532a2b, which is temporarily available from http://www4.in.tum.de/~wenzelm/test/Isabelle_15-Aug-2016.

```
theorem Schroeder-Bernstein: ‹∃ h :: 'a ⇒ 'b. inj h ∧ surj h› if ‹inj f› ‹inj g›
  for f :: ‹'a ⇒ 'b› and g :: ‹'b ⇒ 'a›
proof
  define A where ‹A = lfp (λX. - (g ‘ (- (f ‘ X))))›
  define g' where ‹g' = inv g›
  let ‹?h› = ‹λz. if z ∈ A then f z else g' z›

  have ‹A = - (g ‘ (- (f ‘ A)))›
    unfolding A-def by (rule lfp-unfold) (blast intro: monoI)
  then have A-compl: ‹- A = g ‘ (- (f ‘ A))› by blast
  then have ∗: ‹g' ‘ (- A) = - (f ‘ A)›
    using g'-def ‹inj g› by auto

  show ‹inj ?h ∧ surj ?h›
  proof
    from ∗ show ‹surj ?h› by auto
    have ‹inj-on f A›
      using ‹inj f› by (rule subset-inj-on) blast
    moreover
    have ‹inj-on g' (- A)›
      unfolding g'-def
    proof (rule inj-on-inv-into)
      have ‹g ‘ (- (f ‘ A)) ⊆ range g› by blast
      then show ‹- A ⊆ range g› by (simp only: A-compl)
    qed
    moreover
    have ‹False› if eq: ‹f a = g' b› and a: ‹a ∈ A› and b: ‹b ∈ - A› for a b
    proof -
      from a have fa: ‹f a ∈ f ‘ A› by (rule imageI)
      from b have ‹g' b ∈ g' ‘ (- A)› by (rule imageI)
      with ∗ have ‹g' b ∈ - (f ‘ A)› by simp
      with eq fa show ‹False› by simp
    qed
    ultimately show ‹inj ?h›
      unfolding inj-on-def by (metis ComplI)
  qed
qed
```

Figure 1: Example: Schröder-Bernstein Theorem

double-quotes of Isabelle, where people often ask “What are these odd string literals in the theory source?”

Finally, the use of **define** instead of old **def** hints at unification and clarification of of derived Isar proof elements, notably **obtain** and the newly introduced **consider** element for multi-branch elimination rules.

2 Isar language syntax

The Isar proof language emerges in a bottom-up fashion from *commands* that may be composed according to an interaction *mode* (i.e. "prove", "state", "chain") and implicit nesting of blocks. The resulting language can be described approximatively by a conventional grammar (figure 2). This helps to get an overview, while detailed syntax diagrams are provided for each command in the Isabelle/Isar Reference Manual [3].

Note that this presentation shares auxiliary syntax categories with the manual [3], notably *name*, *vars*, *props*, *thms*, *mixfix*. Multiple items may be repeated a second time, with keyword **and** as separator, but this is ignored in the grammar for simplicity: e.g. "*name*: *props*" should be understood as "*name*: *props* **and** ... **and** *name*: *props*". Names for newly introduced facts or clauses are optional. Contextual annotations, such "**if** *props*", "**for** *vars*", "**fixes** *vars*", "**assumes** *props*", "**premises** *name*" may be omitted.

3 Notable Isar updates

3.1 Structured statements

The main idea is to write Pure rules like $\langle \bigwedge x.\ A\ x \Longrightarrow B\ x \Longrightarrow C\ x \rangle$ in postfix notation $\langle C\ x \rangle$ **if** $\langle A\ x \rangle$ **and** $\langle B\ x \rangle$ **for** x. This emphasizes the outermost Horn-clause structure, and imposes a natural order according to importance: conclusion, premises, parameters (maybe with type constraints).

Structured assumptions may occur in **assume**, but also derived forms like **define** or **obtain**. Here the **if**/**for** notation is turned directly into $\bigwedge/\Longrightarrow$, but vacuous quantifiers are omitted. This is relevant for multiple conclusions, e.g. **assume** $\langle A\ x \rangle$ **and** $\langle B\ y \rangle$ **for** $x\ y$ corresponds to **assume** $\langle \bigwedge x.\ A\ x \rangle$ **and** $\langle \bigwedge y.\ B\ y \rangle$ according to the occurrences of the parameters in the propositions.

Structured conclusions (goals) use the **if**/**for** notation to build an *eigen-context* that fixes the parameters and assumes the premises; the rule structure emerges naturally on export of the result. This corresponds to the well-established notation for toplevel theorem statements **fixes**–**assumes**–**shows**, but due to postfix notation, there is no need for a separate keyword **shows** for the conclusion.

Premises may be named on the spot, e.g. **if** a: $\langle A \rangle$ **and** b: $\langle B \rangle$. The standard name for all premises taken together is "*that*".

Here are some well-known Natural Deduction rules written as structured statements of the command **have**:

- conjunction introduction:
  have $\langle A \wedge B \rangle$ **if** $\langle A \rangle$ **and** $\langle B \rangle$

main = **notepad begin** *statement** **end**
| **theorem** *name*: *props* **if** *name*: *props* **for** *vars*
| **theorem** *name*:
fixes *vars*
assumes *name*: *props*
shows *name*: *props proof*
| **theorem** *name*:
fixes *vars*
assumes *name*: *props*
obtains (*name*) *clause* | ... *proof*

proof = *refinement** *proper-proof*

refinement = **apply** *method*
| **supply** *name* = *thms*
| **subgoal premises** *name* **for** *vars proof*
| **using** *thms*
| **unfolding** *thms*

proper-proof = **proof** *method*$^?$ *statement** **qed** *method*$^?$
| **by** *method method* | **..** | **.** | **sorry** | **done**

statement = **{** *statement** **}** | **next**
| **note** *name* = *thms*
| **let** *term* = *term*
| **write** *name* (*mixfix*)
| **fix** *vars*
| **assume** *name*: *props* **if** *props* **for** *vars*
| **presume** *name*: *props* **if** *props* **for** *vars*
| **define** *clause*
| **case** *name*: *case*
| **then**$^?$ *goal*
| **from** *thms goal*
| **with** *thms goal*
| **also** | **finally** *goal*
| **moreover** | **ultimately** *goal*

goal = **have** *name*: *props* **if** *name*: *props* **for** *vars proof*
| **show** *name*: *props* **if** *name*: *props* **for** *vars proof*
| **show** *name*: *props* **when** *name*: *props* **for** *vars proof*
| **consider** (*name*) *clause* | ... *proof*
| **obtain** (*name*) *clause proof*

clause = *vars* **where** *name*: *props* **if** *props* **for** *vars*

Figure 2: Main grammar of the Isar proof language

- existential introduction:
  have ‹$\exists\, x.\ B\ x$› **if** ‹$B\ a$› **for** a
- disjunction elimination:
  from ‹$A \vee B$› **have** ‹C› **if** ‹$A \Longrightarrow C$› **and** ‹$B \Longrightarrow C$› **for** C
- existential elimination:
  from ‹$\exists\, x.\ B\ x$› **have** ‹C› **if** ‹$\bigwedge x.\ B\ x \Longrightarrow C$› **for** C

Here are concrete proof patterns with more concise reasoning than traditional **fix**–**assume**–**show** and **next** blocks. E.g. iff-introduction works with a minimum of proof blocks like this:

```
have ‹A ⟷ B›
proof
  show ‹B› if ‹A› using that ⟨proof⟩
  show ‹A› if ‹B› using that ⟨proof⟩
qed
```

And this is mathematical induction in the same concise style:

```
have ‹P n› for n :: nat
proof (induct n)
  show ‹P 0› ⟨proof⟩
  show ‹P (Suc n)› if ‹P n› for n using that ⟨proof⟩
qed
```

Since the eigen-context is wrapped tightly around each statement, there is no need to indicate the block structure of sub-proofs via **next** or **{ ... }**.

Weak premises are declared via **presume** instead of **assume**. This goes back to the most ancient stage of Isar, when premises were always weak, i.e. canonical proof decomposition via **fix**–**assume**–**show** did not resolve goal premises by assumption. Since this lead to overcrowded goal states and fragile proofs, it **assume** was later changed to strong premises, and **presume** preserved the old behaviour for conservativity.

For orthogonality, current Isar provides **when** for weak premises (**presume**) and **if** for strong premises (**assume**). Using the new syntax, an old proof pattern for inverted reasoning with **presume** can be expressed as follows:

```
have ‹C›
proof −
  show ?thesis when ‹A› and ‹B›        — eigen-context with weak premises (new in 2016)
    using that by (rule ‹A ⟹ B ⟹ C›)
  show ‹A› ⟨proof⟩
  show ‹B› ⟨proof⟩
qed
```

This proof pattern is not exceedingly exciting, but explicit notation for weak premises for **show** ‹B› **when** ‹A› means that **show** ‹$A \Longrightarrow B$› becomes free for reinterpretation in

the sense of strong premises. This reform of 2016 avoids typical confusion of beginners who experiment with Isar for the first time, and expect that a subgoal $A \Longrightarrow B$ can be solved with **show** ‹$A \Longrightarrow B$›. Consequently, all premises in the subsequent examples are now *strong*:

have ‹$A \longrightarrow B$›
proof
  assume ‹A› — strong premise (since 1999)
  show ‹B› ⟨*proof*⟩
qed

have ‹$A \longrightarrow B$›
proof
  show ‹B› **if** ‹A› — strong premise (new in 2016)
    ⟨*proof*⟩
qed

have ‹$A \longrightarrow B$›
proof
  show ‹$A \Longrightarrow B$› — strong premise (changed in 2016)
    ⟨*proof*⟩
qed

3.2 Elimination statements and cases

Elimination statements express the idea behind $\vee$, $\exists$, $\wedge$ by means of the Isar proof language and the Pure logical framework. With structured statements provided as primitive, it has become easy to introduce variations of the existing **obtain** command.

Multi-branch elimination is expressed by the new **consider** command, which is similar to the existing **theorem**–**obtains** form:

consider (*a*) *x* **where** ‹$A\ x$› | (*b*) *y* **where** ‹$B\ y$› | ...

expands to:

have ‹*thesis*›
  if *a*: ‹$\bigwedge x.\ A\ x \Longrightarrow$ *thesis*› **and** *b*: ‹$\bigwedge y.\ B\ y \Longrightarrow$ *thesis*› **and** ...
  for *thesis*

Here are some well-known Natural Deduction rules written as **consider** statements:

- disjunction elimination:
  from ‹$A \vee B$› **consider** ‹A› | ‹B› **..**
- existential elimination:
  from ‹$\exists x.\ B\ x$› **consider** *x* **where** ‹$B\ x$› **..**

- conjunction elimination:
  from ‹$A \wedge B$› **consider** ‹A› **and** ‹B› **..**

Single-branch elimination still works with the well-known **obtain** command, which may be rephrased via **consider** as follows:

obtain x **where** ‹$A\ x$› $\langle proof \rangle$

expands to:

consider x **where** ‹$A\ x$› $\langle proof \rangle$
fix x **assume** ‹$A\ x$›

As usual in Isar, the assumption is instrumented internally by the resulting **consider** rule ‹$\bigwedge thesis.\ (\bigwedge x.\ A\ x \Longrightarrow thesis) \Longrightarrow thesis$› so that exported results get rid of the obtained assumption (provided the auxiliary parameter x is not mentioned).

Generalized proof by cases uses a multi-branch elimination rule from **consider** with the proof method *cases*, which has been slightly augmented to accept an elimination rule chained into it as method fact. For example:

```
consider (a) A | (b) B | (c) C | (d) D ⟨proof⟩
then have something
proof cases
  case prems: a
  from ‹A› show ?thesis ⟨proof⟩
next
  case prems: b
  from ‹B› show ?thesis ⟨proof⟩
next
  case prems: c
  from ‹C› show ?thesis ⟨proof⟩
next
  case prems: d
  from ‹D› show ?thesis ⟨proof⟩
qed
```

Clausal definitions use the new command **define**, with the same syntax as **obtain**, but more elementary equational logic behind it.

Unlike the old **def** command, it is also possible to present the defining equation in applied form, similar to the **definition** command for theory specifications. This requires an explicit eigen-context with explicit declaration of local variables. For example, **define** f **where** ‹$f\ x = t$› **for** x instead of old **def** $f \equiv$ ‹$\lambda x.\ t$›.

3.3 Structured goal refinements

The syntax category *refinement* in figure 2 admits direct operations on the goal state, before the *proper-proof* is commenced. In practice there are two extremes: no refinement followed by a structured Isar proof, or a list of non-trivial refinements followed by **done** (which is formally a vacuous proof).

The latter case means that the proof degenerates into an "**apply**-script of the form **apply** *method* ... **apply** *method* **done**. Some "improper proof methods" were introduced many years ago, to imitate adhoc access to internal goal parameters (*rule-tac* etc.) and allow easy porting of old tactic scripts.

Isar in 2016 allows to recover proof structure in the middle of unstructured goal refinement. This works via the new **subgoal** command, which is closely related to the existing concept of Isar *goal focus*: a given subgoal $\langle \bigwedge x.\ A\ x \Longrightarrow C\ x \rangle$ is decomposed into context and conclusion, to solve it recursively via **show** $\langle A\ x \Longrightarrow C\ x \rangle$ **for** x (partial focus on parameters) or **show** $\langle C\ x \rangle$ **if** $\langle A\ x \rangle$ **for** x (full focus on premises and parameters).

Schematic variables in the goal state are imported into the focus context as locally *fixed* variables. This means, the nested proof cannot instantiate unknowns. So *logic programming* with synthesized results is not supported by this concept, but it is very rare in practice anyway.

In summary, the **subgoal** command supports the following variants:

- **subgoal** — partial focus with anonymous parameters, i.e. the system invents names for the goal parameters that cannot be accessed in the subsequent proof text;
- **subgoal for** $x\ y$ — partial focus with named parameters (prefix), i.e. the user provides names for some parameters;
- **subgoal for** ... $x\ y$ — partial focus with named parameters (suffix), where the three dots are a literal keyword of the syntax;
- **subgoal premises** *name* — full focus on parameters and premises, which are locally assumed and accessible by the given name (optional). Often the particular name *prems* is used.

Here are abstract proof patterns to illustrate nesting of proper proofs into unstructured goal refinements:

⟨*goal*⟩
subgoal by *method*
subgoal by *method*
done

⟨*goal*⟩
subgoal premises *prems* **for** *x y*
using *prems* ⟨*proof*⟩
subgoal premises *prems* **for** *u v w*
using *prems* ⟨*proof*⟩
done

Of course, the nested proof may again start with an unstructured refinement, according to the syntax of figure 2. This also admits hierarchically structured **apply**-scripts, which are still better than totally unstructured **apply**-scripts.

Moreover note that a **subgoal** proof is subject to parallel processing as any other nested proof of **have**, **show**, **obtain**, **consider** etc.

4 Document preparation and authoring

The ultimate purpose of the Isar proof language is to produce nicely rendered proof documents, with good type-setting quality.

Isabelle document preparation goes back to the early years of Isar (1999): it started out as a simple pretty-printer for Isar theories in LaTeX. Over the years, it has been continuously refined to provide more and more first-class structure managed by Isabelle. This trend also helps the Prover IDE (Isabelle/jEdit) to support the user, see also [2].

In 2016, the main markup commands for explicit document structure are as follows:

- Section headings **chapter**, **section**, **subsection**, **subsubsection**, **paragraph**, **subparagraph**. By default, these are mapped to standard LaTeX macros of the same name. The Prover IDE shows a corresponding tree view of the source; it is also possible to collapse/expand text folds according to that document outline.
- Text blocks via Isar command **text**, **txt**, **text-raw** consisting of paragraphs of text (with slightly different surrounding markup).

Note that these document markup commands no longer require a particular context: they may occur anywhere, even before the initial **theory** header.

Document text is wrapped as *text cartouche*, and often nested as in **text** ‹*abc* ‹*def*› *xyz*›. The structure of the content is as follows.

- Informal words and LaTeX macros, where checking is restricted to spell-checking in the Prover IDE (only for English).
- Special Isabelle control symbols for implicit list structure: itemize, enumerate, description. The notation is similar to Markdown, but uses only paragraphs and lists from its much bigger repertoire[2]. The Prover IDE keeps track of nested lists and paragraphs and highlights that to the user.

[2] http://commonmark.org

- Formal antiquotations that to refer to types, terms, theorems etc. in the text, with formal checking and pretty-printing.

  The classic antiquotation syntax `@{`*name* [*options*] *arguments*`}` may be abbreviated as follows, if the *arguments* consist of at most one text cartouche:

  1. single cartouche argument: `\<^`*name*`>`⟨*argument*⟩, e.g. `\<^emph>`⟨*text*⟩
  2. no argument: `\<^`*name*`>`, e.g. `\<^medskip>`
  3. standard name `cartouche`: ⟨*argument*⟩, e.g. ⟨$x + y$⟩ that is typeset like a formal term, but without formal checking.

  Since `\<^`*name*`>` is notationally an Isabelle control symbol, it can be mapped to some Unicode glyph for rendering. This makes sources visually appealing, without providing full WYSIWYG in the text editor. The manual [4] provides some screenshots in chapter 4. Its own sources[3] may may also serve as an example: the HTML output of Isabelle2016 approximates the PIDE editor view by using the same font, but it lacks formal markup for sub-structures.

There are predefined antiquotations for bold, emphasize, verbatim (type writer font), footnote, with special rendering as Isabelle symbols. This formal notation works better than fragile LaTeX macros: e.g. consider nesting of verbatim text into a footnote.

The new antiquotation called *theory-text* typesets fragments of Isar source (without checking); it is frequently used this document to present snippets of Isar.

5 Summary of relevant NEWS

5.1 Isabelle2016

- Local goals (**have**, **show**) allow structured rule statements with optional eigen-context **if** *prems* **for** *vars*.
- Assumptions (**assume**) allow structured statements **prop if** *prems* **for** *vars* as alternative form for $\bigwedge vars.\ prems \Longrightarrow prop$. Vacuous quantification in assumptions is omitted.
- The meaning of **show** with Pure rule statements has changed: premises are treated strictly in the sense of **assume**. This means, a goal like ⟨$\bigwedge x.\ A\ x \Longrightarrow B\ x \Longrightarrow C\ x$⟩ can be solved completely as follows: **show** ⟨$\bigwedge x.\ A\ x \Longrightarrow B\ x \Longrightarrow C\ x$⟩ or **show** ⟨$C\ x$⟩ **if** ⟨$A\ x$⟩ ⟨$B\ x$⟩ **for** x.
- New command **consider** for generalized elimination and case splitting. This is like a toplevel statement **theorem obtains** used within a proof body; or like a multi-branch **obtain** without activation of the local context elements yet.
- Proof method *cases* allows to specify the rule as first entry of chained facts. This is particularly useful with **consider**.

[3] http://isabelle.in.tum.de/dist/library/Doc/JEdit/JEdit.html

- Command **case** allows fact name and attribute specification: **case** a: $(c\ x\ y)$.
- The standard proof method of commands **proof** and **..** is called *standard* instead of old *default*. Documentation explains **..** more accurately as **by** *standard* instead of **by** *rule*.
- Nesting of Isar goal structure has been clarified: the context after the initial backwards refinement is retained for the whole proof, within all its context sections (as indicated via **next**). This is e.g. relevant for **using**, **including**, **supply**:
- Command **subgoal** allows to impose some structure on backward refinements, to avoid proof scripts degenerating into long of **apply** sequences.
- Command **supply** supports fact definitions during goal refinement (**apply**-scripts).
- Proof method *goal-cases* turns the current subgoals into cases within the context; the conclusion is bound to variable *?case* in each case.

5.2 After Isabelle2016

- Embedded content (e.g. the inner syntax of types, terms, props) may be delimited uniformly via cartouches. This works better than old-fashioned quotes when sublanguages are nested.
- Command **define** f **where** $\langle f\ x = t\rangle$ **for** x supersedes **def** $f \equiv \langle \lambda x.\ t\rangle$.
- Special command `\<proof>` is an alias for **sorry**, with different typesetting: $\langle proof \rangle$.
- Proof methods may refer to the main facts via the dynamic fact *method-facts*. This is particularly useful for Eisbach method definitions.
- Eisbach provides method *use* to modify the main facts of a given method expression, e.g. (*use facts* **in** *simp*) or (*use facts* **in** $\langle$*simp add*: ...$\rangle$).

6 Conclusion

Isar is a living language that continues to unfold and prosper. Many inconveniences and limitations have been settled in the 2016 update. A few are left for future work, in particular:

- Re-unification of attributes *where* and *of* in Eisbach vs. Pure — when this happens Eisbach can be included in Pure.
- Re-unification of attributes *where* and *atomize-elim* as just one attribute, which might be called *compact* and used by default as a wrapper to automated proof methods.

- Re-unification of proof methods *induct* vs. *induction*, and *coinduct* vs. *coinduction*.
- Potential elimination of old tactic emulations, notably *induct-tac*, *case-tac*, or even *rule-tac* and relatives.

Variants of *rule-tac* are typically used to bypass builtin Isar disciplines about use of facts and (restricted) access to goal parameters. With new Eisbach spin-offs like the command **subgoal** or the proof method *use*, there is hope to avoid such legacy tools in the future, but it might turn out infeasible to upgrade existing applications.

Still pending is a major renovation of LaTeX and HTML rendering of Isar proof documents: both should use the rich PIDE document markup for more detailed presentation. HTML output desperately needs better typesetting quality, to get more than just syntax highlighting in the browser.

References

[1] M. Wenzel. Isar — a generic interpretative approach to readable formal proof documents. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics (TPHOLs 1999)*, volume 1690 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[2] M. Wenzel. Asynchronous user interaction and tool integration in Isabelle/PIDE. In G. Klein and R. Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Vienna, Austria*, volume 8558 of *Lecture Notes in Computer Science*. Springer, 2014.

[3] M. Wenzel. *The Isabelle/Isar Reference Manual*, February 2016. http://isabelle.in.tum.de/doc/isar-ref.pdf.

[4] M. Wenzel. *Isabelle/jEdit*, February 2016. http://isabelle.in.tum.de/doc/jedit.pdf.