Isabelle/Isar — a generic framework for human-readable proof documents

Makarius Wenzel

Technische Universität München
Institut für Informatik, Boltzmannstraße 3, 85748 Garching, Germany
http://www.in.tum.de/~wenzelm/

Abstract. Isabelle/Isar is a generic framework for human-readable formal proof documents, both like and unlike Mizar. The Isar proof language provides general principles that may be instantiated to particular object-logics and applications. The design of Isar has emerged from careful analysis of some inherent virtues of the existing logical framework of Isabelle/Pure, notably composition of higher-order natural deduction rules, which is a generalization of Gentzen's original calculus. Thus Isar proof texts may be understood as structured compositions of formal entities of the Pure framework, namely propositions, facts, and goals.
This paper provides an extensive overview of the combined Pure + Isar framework, including a full object-logic definition as a working example. Hereby we hope to illustrate the present stage of our particular journey from insight into the logical framework, to proofs written in Isabelle/Isar.

1 Introduction

1.1 Theory documents

Isabelle/Isar [19, 20, 8, 22] is intended as a generic framework for developing formal mathematical documents with full proof checking. Definitions and proofs are organized as theories; an assembly of theory sources may be presented as a printed document (using PDF-LaTeX). The present paper is an example of the result of processing formally checked sources by the Isabelle/Isar document preparation system.

The main objective of Isar is the design of a human-readable structured proof language, which is called the "primary proof format" in Isar terminology. Such a primary proof language is somewhere in the middle between the extremes of primitive proof objects and actual natural language. In this respect, Isar is a bit more formalistic than Mizar [18, 16, 23, 25], using logical symbols for certain reasoning schemes where Mizar would prefer English words; see [26] for further comparisons of these systems.

So Isar challenges the traditional way of recording informal proofs in mathematical prose, as well as the common tendency to see fully formal proofs directly as objects of some logical calculus (e.g. λ-terms in a version of type theory). In fact, Isar is better understood as an interpreter of a simple block-structured language for describing data flow of local facts and goals, interspersed with occasional invocations of proof methods. Everything is reduced to logical inferences internally, but these steps are somewhat marginal compared to the overall bookkeeping of the interpretation process. Thanks to careful design of the syntax and semantics of Isar language elements, a formal record of Isar instructions may later appear as an intelligible text to the attentive reader.

Isabelle/Isar is based on the existing logical framework of Isabelle/Pure [11, 12], which provides a generic environment for higher-order natural deduction. The approach of generic inference systems in Pure is continued by Isar towards actual proof texts. Concrete applications require another intermediate layer: an object-logic. Isabelle/HOL [9] (simply-typed set-theory) is being used most of the time; Isabelle/ZF is less extensively developed, although it would probably fit better for classical mathematics.

After the pioneering approach of Mizar [18] towards mathematical proofs on the machine has become more widely acknowledged around 1995, some existing interactive theorem provers have been extended to support a similar "declarative mode", such as "a Mizar mode for HOL" [5] or [24]. The design of Isar draws both on Mizar and various Mizar modes, but the resulting Isabelle/Isar is a well-integrated system that continues the original Isabelle tradition. Consequently, Isabelle/Isar is neither another version of Mizar, nor a "declarative mode" for Isabelle. The distinctive style of the logical framework is preserved, and Isar works for all of the usual Isabelle object-logics, such as FOL, ZF, HOL, HOLCF.

This paper is dedicated to elements of structured proofs. We shall briefly give a flavor of plain specifications in Isabelle just now, both axiomatic and definitional ones. For example, in Isabelle/HOL three distinct natural numbers can be postulated like this:

axiomatization $a\ b\ c :: nat$ **where** $a \neq b$ **and** $b \neq c$ **and** $a \neq c$

Stating arbitrary axioms is convenient, but also dangerous. Isabelle methodology follows traditional mathematics in preferring proper definitions. For example:

definition $a :: nat$ **where** $a = 1$
definition $b :: nat$ **where** $b = 2$
definition $c :: nat$ **where** $c = 3$

Definitions like this occasionally have the disadvantage of over-specification: we have introduced particular natural numbers instead of three "arbitrary" ones.

Alternatively we may use the locale mechanism of Isabelle [6, 1], which combines both specification paradigms conveniently: a given axiomatization is wrapped into a predicate definition, subsequent definitions and proofs depend on the resulting context. Viewed from outside, any consequences become relative to explicit assumptions. E.g.

locale *distinct-nats* = **fixes** $a\ b\ c :: nat$ **assumes** $a \neq b$ **and** $b \neq c$ **and** $a \neq c$

Many interesting questions arise from advanced specifications. We shall occasionally return to such issues, when there is an immediate impact on proof composition.

1.2 Proof texts

The Isar proof language offers common principles that are parameterized by entities of the object-logic and application context. This includes declarations for various classes of rules, such as logical introductions / eliminations, and transitivity / symmetry rules. General reasoning mechanisms refer to such declarations from the context, performing "obvious" reasoning steps in a very elementary sense. Moreover, the system admits arbitrarily complex proof methods to be added later, for example specific support for induction proofs [21]. Automated reasoning tools may be integrated as well [14, 15].

In order to illustrate typical natural deduction reasoning in Isar, we assume the background theory and library of Isabelle/HOL [9]. This includes common notions of predicate logic, naive set-theory etc. using fairly standard mathematical notation. From the perspective of generic natural deduction there is nothing special about the logical connectives of HOL ($\wedge$, $\vee$, $\forall$, $\exists$, etc.), only the resulting reasoning principles are relevant to the user. There are similar rules available for set-theory operators ($\cap$, $\cup$, $\bigcap$, $\bigcup$, etc.), or any other theory developed in the library (lattice theory, topology etc.).

Subsequently we briefly review fragments of Isar proof texts corresponding directly to such general natural deduction schemes. The examples shall refer to set-theory.

The following deduction performs $\cap$-introduction, working forwards from assumptions towards the conclusion. We give both the Isar text, and depict the primitive rule involved, as determined by unification of the problem against rules from the context.

assume $x \in A$ **and** $x \in B$
then have $x \in A \cap B$ **..**

$$\frac{x \in A \quad x \in B}{x \in A \cap B}$$

Note that **assume** augments the context, **then** indicates that the current facts shall be used in the next step, and **have** states a local claim. The mysterious "**..**" above is a complete proof of the claim, using the indicated facts and a canonical rule from the context. We could have been more explicit here, writing "**by** (*rule IntI*)" instead.

The format of the $\cap$-introduction rule represents the most basic inference, which proceeds from given premises to a conclusion, without any additional context involved.

The next example performs backwards introduction on $\bigcap \mathcal{A}$, the intersection of all sets within a given set. This requires a nested proof of set membership within a local context of an arbitrary-but-fixed member of the collection:

have $x \in \bigcap \mathcal{A}$
proof
  fix A
  assume $A \in \mathcal{A}$
  show $x \in A$ $\langle proof \rangle$
qed

$$\frac{\begin{array}{c}[A][A \in \mathcal{A}] \\ \vdots \\ x \in A\end{array}}{x \in \bigcap \mathcal{A}}$$

This Isar reasoning pattern again refers to the primitive rule depicted above. The system determines it in the "**proof**" step, which could have been spelt out more explicitly as "**proof** (*rule InterI*)". Note that this rule involves both a local parameter A and an assumption $A \in \mathcal{A}$ in the nested reasoning. This kind of compound rule typically demands a genuine sub-proof in Isar, working backwards rather than forwards as seen before. In the proof body we encounter the **fix**-**assume**-**show** skeleton of nested sub-proofs that is typical for Isar. The final **show** is like **have** followed by an additional refinement of the enclosing claim, using the rule derived from the proof body.

The next example involves $\bigcup \mathcal{A}$, which can be characterized as the set of all x such that $\exists A.\ x \in A \wedge A \in \mathcal{A}$. The elimination rule for $x \in \bigcup \mathcal{A}$ does not mention $\exists$ and $\wedge$ at all, but admits to obtain directly a local A such that $x \in A$ and $A \in \mathcal{A}$ hold. This corresponds to the following Isar proof and inference rule, respectively:

```
assume $x \in \bigcup \mathcal{A}$
then have $C$
proof
  fix $A$
  assume $x \in A$ and $A \in \mathcal{A}$
  show $C$  $\langle proof \rangle$
qed
```

$$\dfrac{x \in \bigcup \mathcal{A} \qquad \begin{array}{c}[A][x \in A, A \in \mathcal{A}]\\ \vdots \\ C\end{array}}{C}$$

Although the Isar proof follows the natural deduction rule closely, the text reads not as natural as anticipated. There is a double occurrence of an arbitrary conclusion C, which represents the final result, but is irrelevant for now. This issue arises for any elimination rule involving local parameters. Isar provides the derived language element **obtain**, which is able to perform the same elimination proof more conveniently:

```
assume $x \in \bigcup \mathcal{A}$
then obtain $A$ where $x \in A$ and $A \in \mathcal{A}$ ..
```

Here we avoid to mention the final conclusion C and return to plain forward reasoning. The rule involved in the "**..**" proof is the same as before.

1.3 Overview

The rest of the paper is structured as follows: §2 reviews the Pure logical framework of Isabelle, §3 covers the main aspects of the Isar proof language, and §4 demonstrates the combined Pure + Isar framework by first-order predicate logic as object-language.

2 The Pure framework

The Pure logic [11, 12] is an intuitionistic fragment of higher-order logic [3]. In type-theoretic parlance, there are three levels of λ-calculus with corresponding arrows: $\Rightarrow$ for syntactic function space (terms depending on terms), $\bigwedge$ for universal quantification (proofs depending on terms), and $\Longrightarrow$ for implication (proofs depending on proofs).

On top of this, Pure implements a generic calculus for nested natural deduction rules, similar to [17]. Here object-logic inferences are internalized as formulae over $\bigwedge$ and $\Longrightarrow$. Combining such rule statements may involve higher-order unification [10].

2.1 Primitive inferences

Term syntax provides explicit notation for abstraction $\lambda x :: \alpha.\ b(x)$ and application $b\ a$, while types are usually implicit thanks to type-inference; terms of type *prop* are called propositions. Logical statements are composed via $\bigwedge x :: \alpha.\ B(x)$ and $A \Longrightarrow B$. Primitive reasoning operates on judgments of the form $\Gamma \vdash \varphi$, with standard introduction and elimination rules for $\bigwedge$ and $\Longrightarrow$ that refer to fixed parameters $x_1, \ldots, x_m$ and hypotheses $A_1, \ldots, A_n$ from the context Γ; the corresponding proof terms are left implicit. The subsequent inference rules define $\Gamma \vdash \varphi$ inductively, relative to a collection of axioms:

$$\frac{(A \text{ axiom})}{\vdash A} \qquad \frac{}{A \vdash A}$$

$$\frac{\Gamma \vdash B(x) \quad x \notin \Gamma}{\Gamma \vdash \bigwedge x.\, B(x)} \qquad \frac{\Gamma \vdash \bigwedge x.\, B(x)}{\Gamma \vdash B(a)}$$

$$\frac{\Gamma \vdash B}{\Gamma - A \vdash A \Longrightarrow B} \qquad \frac{\Gamma_1 \vdash A \Longrightarrow B \quad \Gamma_2 \vdash A}{\Gamma_1 \cup \Gamma_2 \vdash B}$$

Furthermore, Pure provides a built-in equality $\equiv :: \alpha \Rightarrow \alpha \Rightarrow prop$ with axioms for reflexivity, substitution, extensionality, and $\alpha\beta\eta$-conversion on λ-terms.

An object-logic introduces another layer on top of Pure, e.g. with types i for individuals and o for propositions, term constants $Tr :: o \Rightarrow prop$ as (implicit) derivability judgment and connectives like $\wedge :: o \Rightarrow o \Rightarrow o$ or $\forall :: (i \Rightarrow o) \Rightarrow o$, and axioms for object rules such as *conjI*: $A \Longrightarrow B \Longrightarrow A \wedge B$ or *allI*: $(\bigwedge x.\, B\, x) \Longrightarrow \forall x.\, B\, x$. Derived object rules are represented as theorems of Pure.

2.2 Reasoning with rules

Primitive inferences mostly serve foundational purposes. The main reasoning mechanisms of Pure operate on nested natural deduction rules expressed as formulae, using $\bigwedge$ to bind local parameters and $\Longrightarrow$ to express entailment. Multiple parameters and premises are represented by repeating these connectives in a right-associative fashion.

Since $\bigwedge$ and $\Longrightarrow$ commute thanks to $(A \Longrightarrow (\bigwedge x.\, B\, x)) \equiv (\bigwedge x.\, A \Longrightarrow B\, x)$, we may assume w.l.o.g. that rule statements always observe the normal form where quantifiers are pulled in front of implications at each level of nesting. This means that any Pure proposition may be presented as a *Hereditary Harrop Formula* [7] which is of the form $\bigwedge x_1 \ldots x_m.\, H_1 \Longrightarrow \ldots H_n \Longrightarrow A$ for m, $n \geq 0$, and $H_1, \ldots, H_n$ being recursively of the same format, and A atomic. Following the convention that outermost quantifiers are implicit, Horn clauses $A_1 \Longrightarrow \ldots A_n \Longrightarrow A$ are a special case of this.

Goals are also represented as rules: $A_1 \Longrightarrow \ldots A_n \Longrightarrow C$ states that the sub-goals $A_1, \ldots, A_n$ entail the result C; for $n = 0$ the goal is finished. To allow C being a rule statement itself, we introduce the protective marker $\# :: prop \Rightarrow prop$, which is defined as identity and hidden from the user. We initialize and finish goal states as follows:

$$\frac{}{C \Longrightarrow \#C}\ (init) \qquad \frac{\#C}{C}\ (finish)$$

Goal states are refined in intermediate proof steps until a finished form is achieved. Here the two main reasoning principles are *resolve*, for back-chaining a rule against a sub-goal (replacing it by zero or more sub-goals), and *close*, for solving a sub-goal (finding a short-circuit with local assumptions). Below $\overline{x}$ stands for $x_1, \ldots, x_n$ $(n \geq 0)$.

$$\frac{\begin{array}{r@{\,}l} rule\!: & \overline{A}\; \overline{a} \Longrightarrow B\; \overline{a} \\ goal\!: & (\bigwedge \overline{x}.\; \overline{H}\; \overline{x} \Longrightarrow B'\; \overline{x}) \Longrightarrow C \\ goal\ unifier\!: & (\lambda \overline{x}.\; B\; (\overline{a}\; \overline{x}))\; \theta = B'\theta \end{array}}{(\bigwedge \overline{x}.\; \overline{H}\; \overline{x} \Longrightarrow \overline{A}\; (\overline{a}\; \overline{x}))\; \theta \Longrightarrow C\, \theta}\ (resolve)$$

$$\frac{\begin{array}{r}\textit{goal}\colon (\bigwedge \overline{x}.\ \overline{H}\ \overline{x} \Longrightarrow A\ \overline{x}) \Longrightarrow C \\ \textit{assm unifier}\colon A\,\theta = H_i\,\theta\ \ \text{(for some } H_i\text{)}\end{array}}{C\,\theta}\ (\textit{close})$$

The following trace illustrates goal-oriented reasoning in Isabelle/Pure:

$$\begin{array}{rl}
(A \wedge B \Longrightarrow B \wedge A) \Longrightarrow \#(A \wedge B \Longrightarrow B \wedge A) & (\textit{init}) \\
(A \wedge B \Longrightarrow B) \Longrightarrow (A \wedge B \Longrightarrow A) \Longrightarrow \#\ldots & (\textit{resolve}\ B \Longrightarrow A \Longrightarrow B \wedge A) \\
(A \wedge B \Longrightarrow A \wedge B) \Longrightarrow (A \wedge B \Longrightarrow A) \Longrightarrow \#\ldots & (\textit{resolve}\ A \wedge B \Longrightarrow B) \\
(A \wedge B \Longrightarrow A) \Longrightarrow \#\ldots & (\textit{close}) \\
(A \wedge B \Longrightarrow B \wedge A) \Longrightarrow \#\ldots & (\textit{resolve}\ A \wedge B \Longrightarrow A) \\
\#\ldots & (\textit{close}) \\
A \wedge B \Longrightarrow B \wedge A & (\textit{finish})
\end{array}$$

Compositions of *close* after *resolve* occurs quite often, typically in elimination steps. Traditional Isabelle tactics accommodate this by a combined *elim-resolve* principle. In contrast, Isar uses a slightly more refined combination, where the assumptions to be closed are marked explicitly, using again the protective marker #:

$$\frac{\begin{array}{r}\textit{sub-proof}\colon \overline{G}\ \overline{a} \Longrightarrow B\ \overline{a} \\ \textit{goal}\colon (\bigwedge \overline{x}.\ \overline{H}\ \overline{x} \Longrightarrow B'\ \overline{x}) \Longrightarrow C \\ \textit{goal unifier}\colon (\lambda \overline{x}.\ B\ (\overline{a}\ \overline{x}))\,\theta = B'\theta \\ \textit{assm unifiers}\colon (\lambda \overline{x}.\ G_j\ (\overline{a}\ \overline{x}))\,\theta = \#H_i\,\theta\ \ \text{(for each marked } G_j \text{ some } \#H_i\text{)}\end{array}}{(\bigwedge \overline{x}.\ \overline{H}\ \overline{x} \Longrightarrow \overline{G}'\ (\overline{a}\ \overline{x}))\,\theta \Longrightarrow C\,\theta}\ (\textit{refine})$$

Here the *sub-proof* rule stems from the main **fix**-**assume**-**show** skeleton of Isar (cf. §3.3): each assumption indicated in the text results in a marked premise G above.

3 The Isar proof language

Structured proofs are presented as high-level expressions for composing entities of Pure (propositions, facts, and goals). The Isar proof language allows to organize reasoning within the underlying rule calculus of Pure, but Isar is not another logical calculus!

Isar is an exercise in sound minimalism. Approximately half of the language is introduced as primitive, the rest defined as derived concepts. The following grammar describes the core language (category *proof*), which is embedded into theory specification elements such as **theorem**; see also §3.2 for the separate category *statement*.

theory-element = **theorem** *statement proof* | **definition** ... | ...
proof = *prefix** **proof** *method*$^?$ *element** **qed** *method*$^?$
prefix = **using** *facts* **and** ...
| **unfolding** *facts* **and** ...

$$
\begin{array}{rcl}
\mathit{element} & = & \textbf{\{}\ \mathit{element}^*\ \textbf{\}} \\
& | & \textbf{next} \\
& | & \textbf{note}\ \mathit{name} = \mathit{facts}\ \textbf{and} \ldots \\
& | & \textbf{let}\ \mathit{pattern} = \mathit{term}\ \textbf{and} \ldots \\
& | & \textbf{fix}\ \mathit{vars}\ \textbf{and} \ldots \\
& | & \textbf{assm}\ \ll\!\mathit{rule}\!\gg\ \mathit{name}\colon \mathit{props}\ \textbf{and} \ldots \\
& | & \textbf{then}^?\ \mathit{claim} \\
\mathit{claim} & = & \textbf{have}\ \mathit{name}\colon \mathit{props}\ \textbf{and} \ldots \mathit{proof} \\
& | & \textbf{show}\ \mathit{name}\colon \mathit{props}\ \textbf{and} \ldots \mathit{proof}
\end{array}
$$

Here the **and** keyword separates simultaneous elements of the same kind.

The syntax for terms and propositions is inherited from Pure (and the object-logic). A *pattern* is a *term* with schematic variables, to be bound by higher-order matching.

Facts may be referenced by name or proposition. E.g. the result of "**have** *a*: *A* ⟨*proof*⟩" becomes available both as *a* and ‹*A*›. Moreover, fact expressions may involve attributes that modify either the theorem or the background context. For example, the expression "*a* [*OF b*]" refers to the composition of two facts according to the *resolve* inference of §2.2, while "*a* [*intro*]" declares a fact as introduction rule in the context.

The special name "*this*" always refers to the last result, as produced by **note**, **assm**, **have**, or **show**. Since **note** occurs frequently together with **then** we provide some abbreviations: "**from** *a*" for "**note** *a* **then**", and "**with** *a*" for "**from** *a* **and** *this*".

The *method* category is essentially a parameter and may be populated later. Methods use the facts indicated by **then** or **using**, and then operate on the goal state. Some basic methods are predefined: "−" leaves the goal unchanged, "*this*" applies the facts as rules to the goal, "*rule*" applies the facts to another rule and the result to the goal (both "*this*" and "*rule*" refer to *resolve* of §2.2). The secondary arguments to "*rule*" may be specified explicitly as in "(*rule a*)", or picked from the context. In the latter case, the system first tries rules declared as [*elim*] or [*dest*], followed by those declared as [*intro*].

The default method for **proof** is "*rule*" (arguments picked from the context), for **qed** it is "−". Further abbreviations for terminal proof steps are "**by** *method*$_1$ *method*$_2$" for "**proof** *method*$_1$ **qed** *method*$_2$", and "**..**" for "**by** *rule*", and "**.**" for "**by** *this*". The **unfolding** element operates directly on the current facts and goal by applying equalities.

Block structure can be indicated explicitly by "**{** . . . **}**", although the body of a sub-proof already involves implicit nesting. In any case, **next** jumps into the next section of a block, i.e. it acts like closing an implicit block scope and opening another one.

The remaining elements **fix** and **assm** build up a local context (see §3.1), while **show** refines a pending sub-goal by the rule resulting from a nested sub-proof (see §3.3). Further derived concepts will support calculational reasoning (see §3.4).

3.1 Context elements

In judgments $\Gamma \vdash \varphi$ of the primitive framework, Γ essentially acts like a proof context. Isar elaborates this idea towards a higher-level notion, with separate information for type-inference, term abbreviations, local facts, hypotheses etc.

The element **fix** $x :: \alpha$ declares a local parameter, i.e. an arbitrary-but-fixed entity of a given type; in results exported from the context, x may become anything. The **assm** element provides a general interface to hypotheses: "**assm** $\ll rule \gg A$" produces $A \vdash A$ locally, while the included inference rule tells how to discharge A from results $A \vdash B$ later on. There is no user-syntax for $\ll rule \gg$, i.e. **assm** may only occur in derived elements that provide a suitable inference internally. In particular, "**assume** A" abbreviates "**assm** $\ll discharge\# \gg A$", and "**def** $x \equiv a$" abbreviates "**fix** x **assm** $\ll expand \gg x \equiv a$", involving the following inferences:

$$\frac{\Gamma \vdash B}{\Gamma - A \vdash \#A \Longrightarrow B}\ (discharge\#) \qquad \frac{\Gamma \vdash B\ x}{\Gamma - (x \equiv a) \vdash B\ a}\ (expand)$$

The most interesting derived element in Isar is **obtain** [20, §5.3], which supports generalized elimination steps in a purely forward manner. This is similar to `consider` in Mizar [18, 16, 23, 25], although the way to get there is quite different, and the result is not tied to particular notions of predicate logic (such as $\wedge$, $\exists$).

The **obtain** element takes a specification of parameters $\overline{x}$ and assumptions $\overline{A}$ to be added to the context, together with a proof of a reduction rule stating that this extension is conservative (i.e. may be removed from closed results later on):

```
⟨facts⟩ obtain $\overline{x}$ where $\overline{A}\ \overline{x}$ ⟨proof⟩ ≡
  have reduction: $\bigwedge thesis.\ (\bigwedge \overline{x}.\ \overline{A}\ \overline{x} \Longrightarrow thesis) \Longrightarrow thesis$⟩
  proof −
    fix thesis
    assume [intro]: $\bigwedge \overline{x}.\ \overline{A}\ \overline{x} \Longrightarrow thesis$
    show thesis using ⟨facts⟩ ⟨proof⟩
  qed
  fix $\overline{x}$ assm ≪eliminate reduction≫ $\overline{A}\ \overline{x}$
```

$$\frac{\begin{array}{l} reduction\colon \Gamma \vdash \bigwedge thesis.\ (\bigwedge \overline{x}.\ \overline{A}\ \overline{x} \Longrightarrow thesis) \Longrightarrow thesis \\ \quad result\colon \Gamma \cup \overline{A}\ \overline{y} \vdash B \end{array}}{\Gamma \vdash B}\ (\mathsf{eliminate})$$

Here the name "*thesis*" is a specific convention for an arbitrary-but-fixed proposition; in the primitive natural deduction rules shown before we have occasionally used C. The whole statement of "**obtain** x **where** $A\ x$" may be read as a claim that $A\ x$ may be assumed for some arbitrary-but-fixed x. Also note that "**obtain** A **and** B" without parameters is similar to "**have** A **and** B", but the latter involves multiple sub-goals.

The subsequent Isar proof texts explain all context elements introduced above using the formal proof language itself. After finishing a local proof within a block, we indicate the exported result via **note**. This illustrates the meaning of Isar context elements without goals getting in between.

```
{
  fix x
  have B x
    ⟨proof⟩
}
note ⟨⋀x. B x⟩
```

```
{
  def x ≡ a
  have B x
    ⟨proof⟩
}
note ⟨B a⟩
```

```
{
  assume A
  have B
    ⟨proof⟩
}
note ⟨A ⟹ B⟩
```

```
{
  obtain x
    where A x  ⟨proof⟩
  have B  ⟨proof⟩
}
note ⟨B⟩
```

3.2 Structured statements

The category *statement* of top-level theorem specifications is defined as follows:

$$\begin{aligned}
\mathit{statement} &\equiv \mathit{name}\colon \mathit{props}\ \mathbf{and}\ \ldots \\
&\mid \mathit{context}^{*}\ \mathit{conclusion} \\
\mathit{context} &\equiv \mathbf{fixes}\ \mathit{vars}\ \mathbf{and}\ \ldots \\
&\mid \mathbf{assumes}\ \mathit{name}\colon \mathit{props}\ \mathbf{and}\ \ldots \\
\mathit{conclusion} &\equiv \mathbf{shows}\ \mathit{name}\colon \mathit{props}\ \mathbf{and}\ \ldots \\
&\mid \mathbf{obtains}\ \mathit{vars}\ \mathbf{and}\ \ldots\ \mathbf{where}\ \mathit{name}\colon \mathit{props}\ \mathbf{and}\ \ldots\ \ \mathsf{|}\ \ \ldots
\end{aligned}$$

A simple *statement* consists of named propositions. The full form admits local context elements followed by the actual conclusions, such as “**fixes** *x* **assumes** *A x* **shows** *B x*”. The final result emerges as a Pure rule after discharging the context: $\bigwedge x.\ A\ x \Longrightarrow B\ x$.

The **obtains** variant is another abbreviation defined below; unlike **obtain** (cf. §3.1) there may be several “cases” separated by “|”, each consisting of several parameters (*vars*) and several premises (*props*). This specifies multi-branch elimination rules.

$$\begin{aligned}
&\mathbf{obtains}\ \overline{x}\ \mathbf{where}\ \overline{A}\ \overline{x}\ \ \mathsf{|}\ \ \ldots\ \equiv \\
&\quad \mathbf{fixes}\ \mathit{thesis} \\
&\quad \mathbf{assumes}\ [\mathit{intro}]\colon \bigwedge\overline{x}.\ \overline{A}\ \overline{x} \Longrightarrow \mathit{thesis}\ \ \mathbf{and}\ \ \ldots \\
&\quad \mathbf{shows}\ \mathit{thesis}
\end{aligned}$$

Presenting structured statements in such an “open” format usually simplifies the subsequent proof, because the outer structure of the problem is already laid out directly. E.g. consider the following canonical patterns for **shows** and **obtains**, respectively:

```
theorem
  fixes x and y
  assumes A x and B y
  shows C x y
proof −
  from ‹A x› and ‹B y›
  show C x y ⟨proof⟩
qed
```

```
theorem
  obtains x and y
  where A x and B y
proof −
  have A a and B b ⟨proof⟩
  then show thesis ..
qed
```

Here local facts ‹*A x*› and ‹*B y*› are referenced immediately; there is no need to decompose the logical rule structure again. In the second proof the final “**then show** *thesis* **..**” involves the local reduction rule $\bigwedge x\ y.\ A\ x \Longrightarrow B\ y \Longrightarrow \mathit{thesis}$ for the particular instance of terms *a* and *b* produced in the body.

3.3 Structured proof refinement

By breaking up the grammar for the Isar proof language, we may understand a proof text as a linear sequence of individual proof commands. These are interpreted as transitions of the Isar virtual machine (Isar/VM), which operates on a block-structured configuration in single steps. This allows users to write proof texts in an incremental manner, and inspect intermediate configurations for debugging.

The basic idea is analogous to evaluating algebraic expressions on a stack machine: $(a + b) \cdot c$ then corresponds to a sequence of single transitions for each symbol $($, a, $+$, b, $)$, $\cdot$, c. In Isar the algebraic values are facts or goals, and the operations are inferences.

The Isar/VM state maintains a stack of nodes, each node contains the local proof context, the linguistic mode, and a pending goal (optional). The mode determines the type of transition that may be performed next, it essentially alternates between forward and backward reasoning. For example, in *state* mode Isar acts like a mathematical scratch-pad, accepting declarations like **fix**, **assume**, and claims like **have**, **show**. A goal statement changes the mode to *prove*, which means that we may now refine the problem via **unfolding** or **proof**. Then we are again in *state* mode of a proof body, which may issue **show** statements to solve pending sub-goals. A concluding **qed** will return to the original *state* mode one level upwards. The subsequent Isar/VM trace indicates block structure, linguistic mode, goal state, and inferences:

have $A \longrightarrow B$	*begin*	*prove*	$(A \longrightarrow B) \Longrightarrow \#(A \longrightarrow B)$	(init)
proof		*state*	$(A \Longrightarrow B) \Longrightarrow \#(A \longrightarrow B)$	$(\mathit{resolve}\ (A \Longrightarrow B) \Longrightarrow A \longrightarrow B)$
assume A		*state*		
show B	*begin*	*prove*		
$\langle \mathit{proof} \rangle$	*end*	*state*	$\#(A \longrightarrow B)$	$(\mathit{refine}\ \#A \Longrightarrow B)$
qed	*end*	*state*	$A \longrightarrow B$	(finish)

Here the *refine* inference from §2.2 mediates composition of Isar sub-proofs nicely. Observe that this principle incorporates some degree of freedom in proof composition. In particular, the proof body allows parameters and assumptions to be re-ordered, or commuted according to Hereditary Harrop Form. Moreover, context elements that are not used in a sub-proof may be omitted altogether. For example:

have $\bigwedge x\, y.\ A\ x \Longrightarrow B\ y \Longrightarrow C\ x\ y$
proof $-$
  fix x **and** y
  assume $A\ x$ **and** $B\ y$
  show $C\ x\ y$ $\langle \mathit{proof} \rangle$
qed

have $\bigwedge x\, y.\ A\ x \Longrightarrow B\ y \Longrightarrow C\ x\ y$
proof $-$
  fix y **assume** $B\ y$
  fix x **assume** $A\ x$
  show $C\ x\ y$ $\langle \mathit{proof} \rangle$
qed

have $\bigwedge x\, y.\ A\ x \Longrightarrow B\ y \Longrightarrow C\ x\ y$
proof $-$
  fix x **assume** $A\ x$
  fix y **assume** $B\ y$
  show $C\ x\ y$ $\langle \mathit{proof} \rangle$
qed

have $\bigwedge x\, y.\ A\ x \Longrightarrow B\ y \Longrightarrow C\ x\ y$
proof $-$
  fix y **assume** $B\ y$
  fix x
  show $C\ x\ y$ $\langle \mathit{proof} \rangle$
qed

Such "peephole optimizations" of Isar texts are practically important to improve readability, by rearranging contexts elements according to the natural flow of reasoning in the body, while still observing the overall scoping rules.

This illustrates the basic idea of structured proof processing in Isar. The main mechanisms are based on natural deduction rule composition within the Pure framework. In particular, there are no direct operations on goal states within the proof body. Moreover, there is no hidden automated reasoning involved, just plain unification.

3.4 Calculational reasoning

The present Isar infrastructure is sufficiently flexible to support calculational reasoning (chains of transitivity steps) as derived concept. The generic proof elements introduced below depend on rules declared as $[trans]$ in the context. It is left to the object-logic to provide a suitable rule collection for mixed $=$, $<$, $\leq$, $\subset$, $\subseteq$ etc. Due to the flexibility of rule composition (§2.2), substitution of equals by equals is covered as well, even substitution of inequalities involving monotonicity conditions; see also [20, §6] and [2].

The generic calculational mechanism is based on the observation that rules such as $x = y \Longrightarrow y = z \Longrightarrow x = z$ proceed from the premises towards the conclusion in a deterministic fashion. Thus we may reason in forward mode, feeding intermediate results into rules selected from the context. The course of reasoning is organized by maintaining a secondary fact called "*calculation*", apart from the primary "*this*" already provided by the Isar primitives. In the definitions below, *OF* is *resolve* (§2.2) with multiple rule arguments, and *trans* refers to a suitable rule from the context:

$$\begin{aligned} \textbf{also}_0 &\equiv \textbf{note}\ \mathit{calculation} = \mathit{this} \\ \textbf{also}_{n+1} &\equiv \textbf{note}\ \mathit{calculation} = \mathit{trans}\ [\mathit{OF}\ \mathit{calculation}\ \mathit{this}] \\ \textbf{finally} &\equiv \textbf{also from}\ \mathit{calculation} \end{aligned}$$

The start of a calculation is determined implicitly in the text: here **also** sets *calculation* to the current result; any subsequent occurrence will update *calculation* by combination with the next result and a transitivity rule. The calculational sequence is concluded via **finally**, where the final result is exposed for use in a concluding claim.

Here is a canonical proof pattern, using **have** to establish the intermediate results:

have $a = b$ $\langle proof \rangle$
also have $\ldots = c$ $\langle proof \rangle$
also have $\ldots = d$ $\langle proof \rangle$
finally have $a = d$ **.**

The term "..." above is a special abbreviation provided by the Isabelle/Isar syntax layer: it statically refers to the right-hand side argument of the previous statement given in the text. Thus it happens to coincide with relevant sub-expressions in the calculational chain, but the exact correspondence is dependent on the transitivity rules being involved.

Symmetry rules such as $x = y \Longrightarrow y = x$ are like transitivities with only one premise. Isar maintains a separate rule collection declared via $[sym]$, to be used in fact expressions "a $[symmetric]$", or single-step proofs "**assume** $x = y$ **then have** $y = x$ **..**".

4 Example: First-order Predicate Logic

In order to commence a new object-logic within Isabelle/Pure we introduce abstract syntactic categories i for individuals and o for object-propositions. The latter is embedded into the language of Pure propositions by means of a separate judgment.

typedecl i
typedecl o

```
judgment
  Tr :: o ⇒ prop   (- 5)
```

Note that the object-logic judgement is implicit in the syntax: writing *A* produces *Tr A* internally. From the Pure perspective this means "*A* is derivable in the object-logic".

4.1 Equational reasoning

Equality is axiomatized as a binary predicate on individuals, with reflexivity as introduction, and substitution as elimination principle. Note that the latter is particularly convenient in a framework like Isabelle, because syntactic congruences are implicitly produced by unification of *B x* against expressions containing occurrences of *x*.

```
axiomatization
  equal :: i ⇒ i ⇒ o  (infix = 50)
where
  refl [intro]: x = x and
  subst [elim]: x = y ⟹ B x ⟹ B y
```

Substitution is very powerful, but also hard to control in full generality. We derive some common symmetry / transitivity schemes of as particular consequences.

```
theorem sym [sym]:
  assumes x = y
  shows y = x
proof −
  have x = x ..
  with ⟨x = y⟩ show y = x ..
qed

theorem forw-subst [trans]:
  assumes y = x and B x
  shows B y
proof −
  from ⟨y = x⟩ have x = y ..
  from this and ⟨B x⟩ show B y ..
qed

theorem back-subst [trans]:
  assumes B x and x = y
  shows B y
proof −
  from ⟨x = y⟩ and ⟨B x⟩
  show B y ..
qed

theorem trans [trans]:
  assumes x = y and y = z
  shows x = z
proof −
```

```
  from ⟨$y = z$⟩ and ⟨$x = y$⟩
  show $x = z$ ..
qed
```

4.2 Group theory

As an example for equational reasoning we consider some bits of group theory. The subsequent locale definition postulates group operations and axioms; we also derive some consequences of this specification.

```
locale group =
  fixes prod :: $i \Rightarrow i \Rightarrow i$  (infix $\circ$ 70)
    and inv :: $i \Rightarrow i$  ((-$^{-1}$) [1000] 999)
    and unit :: $i$  ($1$)
  assumes assoc: $(x \circ y) \circ z = x \circ (y \circ z)$
    and left-unit:  $1 \circ x = x$
    and left-inv: $x^{-1} \circ x = 1$
begin

theorem right-inv: $x \circ x^{-1} = 1$
proof −
  have $x \circ x^{-1} = 1 \circ (x \circ x^{-1})$ by (rule left-unit [symmetric])
  also have $\ldots = (1 \circ x) \circ x^{-1}$ by (rule assoc [symmetric])
  also have $1 = (x^{-1})^{-1} \circ x^{-1}$ by (rule left-inv [symmetric])
  also have $\ldots \circ x = (x^{-1})^{-1} \circ (x^{-1} \circ x)$ by (rule assoc)
  also have $x^{-1} \circ x = 1$ by (rule left-inv)
  also have $((x^{-1})^{-1} \circ \ldots) \circ x^{-1} = (x^{-1})^{-1} \circ (1 \circ x^{-1})$ by (rule assoc)
  also have $1 \circ x^{-1} = x^{-1}$ by (rule left-unit)
  also have $(x^{-1})^{-1} \circ \ldots = 1$ by (rule left-inv)
  finally show $x \circ x^{-1} = 1$ .
qed

theorem right-unit: $x \circ 1 = x$
proof −
  have $1 = x^{-1} \circ x$ by (rule left-inv [symmetric])
  also have $x \circ \ldots = (x \circ x^{-1}) \circ x$ by (rule assoc [symmetric])
  also have $x \circ x^{-1} = 1$ by (rule right-inv)
  also have $\ldots \circ x = x$ by (rule left-unit)
  finally show $x \circ 1 = x$ .
qed
```

Reasoning from basic axioms is notoriously tedious. Our proofs work by producing various instances of the given rules (potentially the symmetric form) using the pattern "**have** *eq* **by** (*rule r*)" and composing the chain of results via **also**/**finally**. These steps may involve any of the transitivity rules declared in §4.1, namely *trans* in combining the first two results in *right-inv* and in the final steps of both proofs, *forw-subst* in the first combination of *right-unit*, and *back-subst* in all other calculational steps.

Occasional substitutions in calculations are adequate, but should not be over-emphasized. The other extreme is to compose a chain by plain transitivity only, with replacements occurring always in topmost position. For example:

have $x \circ 1 = x \circ (x^{-1} \circ x)$ **unfolding** *left-inv* **..**
also have $\ldots = (x \circ x^{-1}) \circ x$ **unfolding** *assoc* **..**
also have $\ldots = 1 \circ x$ **unfolding** *right-inv* **..**
also have $\ldots = x$ **unfolding** *left-unit* **..**
finally have $x \circ 1 = x$ **.**

Here we have re-used the built-in mechanism for unfolding definitions in order to normalize each equational problem. A more realistic object-logic would include proper setup for the Simplifier, the main automated tool for equational reasoning in Isabelle. Then "**unfolding** *left-inv* **..**" would become "**by** (*simp add*: *left-inv*)" etc.

end

4.3 Propositional logic

We axiomatize basic connectives of propositional logic: implication, disjunction, and conjunction. The associated rules are modeled after Gentzen's natural deduction [4].

axiomatization
imp :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\longrightarrow$ 25) **where**
impI [*intro*]: $(A \Longrightarrow B) \Longrightarrow A \longrightarrow B$ **and**
impD [*dest*]: $(A \longrightarrow B) \Longrightarrow A \Longrightarrow B$

axiomatization
disj :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\vee$ 30) **where**
disjE [*elim*]: $A \vee B \Longrightarrow (A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C$ **and**
disjI$_1$ [*intro*]: $A \Longrightarrow A \vee B$ **and**
disjI$_2$ [*intro*]: $B \Longrightarrow A \vee B$

axiomatization
conj :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\wedge$ 35) **where**
conjI [*intro*]: $A \Longrightarrow B \Longrightarrow A \wedge B$ **and**
conjD$_1$ [*dest*]: $A \wedge B \Longrightarrow A$ **and**
conjD$_2$ [*dest*]: $A \wedge B \Longrightarrow B$

The conjunctive destructions have the disadvantage that decomposing $A \wedge B$ involves an immediate decision which component should be projected. The more convenient simultaneous elimination $A \wedge B \Longrightarrow (A \Longrightarrow B \Longrightarrow C) \Longrightarrow C$ can be derived as follows:

theorem *conjE* [*elim*]:
assumes $A \wedge B$
obtains A **and** B
proof
from $\langle A \wedge B \rangle$ **show** A **..**
from $\langle A \wedge B \rangle$ **show** B **..**
qed

Here is an example of swapping conjuncts with a single intermediate elimination step:

assume $A \wedge B$
then obtain B **and** A **..**
then have $B \wedge A$ **..**

Note that the analogous elimination for disjunction "**assumes** $A \vee B$ **obtains** $A \mid B$" coincides with the original axiomatization of *disjE*.

We continue propositional logic by introducing absurdity with its characteristic elimination. Plain truth may then be defined as a proposition that is trivially true.

```
axiomatization
  false :: o  (⊥) where
  falseE [elim]: ⊥ ⟹ A

definition
  true :: o  (⊤) where
  ⊤ ≡ ⊥ → ⊥

theorem trueI [intro]: ⊤
  unfolding true-def ..
```

Now negation represents an implication towards absurdity:

```
definition
  not :: o ⇒ o  (¬ - [40] 40) where
  ¬ A ≡ A → ⊥

theorem notI [intro]:
  assumes A ⟹ ⊥
  shows ¬ A
unfolding not-def
proof
  assume A
  then show ⊥ by (rule ‹A ⟹ ⊥›)
qed

theorem notE [elim]:
  assumes ¬ A and A
  shows B
proof −
  from ‹¬ A› have A → ⊥ unfolding not-def .
  from ‹A → ⊥› and ‹A› have ⊥ ..
  then show B ..
qed
```

4.4 Classical logic

Subsequently we state the principle of classical contradiction as a local assumption. Thus we refrain from forcing the object-logic into the classical perspective. Within that context, we may derive well-known consequences of the classical principle.

```
locale classical =
  assumes classical: (¬ C ⟹ C) ⟹ C
begin
```

```
theorem double-negation:
  assumes ¬ ¬ C
  shows C
proof (rule classical)
  assume ¬ C
  with ‹¬ ¬ C› show C ..
qed

theorem tertium-non-datur: C ∨ ¬ C
proof (rule double-negation)
  show ¬ ¬ (C ∨ ¬ C)
  proof
    assume ¬ (C ∨ ¬ C)
    have ¬ C
    proof
      assume C then have C ∨ ¬ C ..
      with ‹¬ (C ∨ ¬ C)› show ⊥ ..
    qed
    then have C ∨ ¬ C ..
    with ‹¬ (C ∨ ¬ C)› show ⊥ ..
  qed
qed
```

These examples illustrate both classical reasoning and non-trivial propositional proofs in general. All three rules characterize classical logic independently, but the original rule is already the most convenient to use, because it leaves the conclusion unchanged. Note that $(\neg\, C \Longrightarrow C) \Longrightarrow C$ fits again into our format for eliminations, despite the additional twist that the context refers to the main conclusion. So we may write *classical* as the Isar statement "**obtains** ¬ *thesis*". This also explains nicely how classical reasoning really works: whatever the main *thesis* might be, we may always assume its negation!

end

4.5 Quantifiers

Representing quantifiers is easy, thanks to the higher-order nature of the underlying framework. According to the well-known technique introduced by Church [3], quantifiers are operators on predicates, which are syntactically represented as λ-terms of type $i \Rightarrow o$. Specific binder notation relates $All\ (\lambda x.\ B\ x)$ to $\forall\, x.\ B\ x$ etc.

```
axiomatization
  All :: (i ⇒ o) ⇒ o  (binder ∀ 10) where
  allI [intro]: (⋀x. B x) ⟹ ∀ x. B x and
  allD [dest]: (∀ x. B x) ⟹ B a

axiomatization
  Ex :: (i ⇒ o) ⇒ o  (binder ∃ 10) where
  exI [intro]: B a ⟹ (∃ x. B x) and
  exE [elim]: (∃ x. B x) ⟹ (⋀x. B x ⟹ C) ⟹ C
```

The *exE* rule corresponds to an Isar statement "**assumes** ∃ *x*. *B x* **obtains** *x* **where** *B x*". In the following example we illustrate quantifier reasoning with all four rules:

```
theorem
  assumes ∃ x. ∀ y. R x y
  shows ∀ y. ∃ x. R x y
proof    — ∀ introduction
  obtain x where ∀ y. R x y using ⟨∃ x. ∀ y. R x y⟩ ..    — ∃ elimination
  fix y have R x y using ⟨∀ y. R x y⟩ ..    — ∀ destruction
  then show ∃ x. R x y ..    — ∃ introduction
qed
```

4.6 Further definitions

Real applications routinely work with derived concepts defined in terms of existing principles. This is illustrated below by an artificial notion of *frob P Q* for logical predicates *P* and *Q*, where *frob P Q* holds whenever *P x* and *Q y* hold for some *x* and *y*. The most basic definition rephrases this idea in terms of connectives of predicate logic:

```
definition frob P Q ≡ ∃ x y. P x ∧ Q y
```

Reasoning with *frob P Q* demands characteristic rules, which are derived as follows:

```
theorem frobI [intro]:
  assumes P x and Q y
  shows frob P Q
proof −
  from ⟨P x⟩ and ⟨Q y⟩ have P x ∧ Q y ..
  then have ∃ y. P x ∧ Q y ..
  then have ∃ x y. P x ∧ Q y ..
  then show frob P Q unfolding frob-def .
qed

theorem frobE [elim]:
  assumes frob P Q
  obtains x and y where P x and Q y
proof −
  from ⟨frob P Q⟩ have ∃ x y. P x ∧ Q y unfolding frob-def .
  then obtain x where ∃ y. P x ∧ Q y ..
  then obtain y where P x ∧ Q y ..
  then obtain P x and Q y ..
  then show thesis ..
qed
```

Now we can use *frob* in applications, without appealing to the primitive definition again:

```
  assume P a and Q b
  then have frob P Q ..

  assume frob P Q
  then obtain x y where P x and Q y ..
```

The above transformation of a definition into canonical reasoning patterns works, but is still somewhat cumbersome. This is typically the place to ask for automated reasoning support to deduce *frobI* and *frobE* from *frob-def* in a single invocation each. For example, the following typically appears in applications of Isabelle/HOL [9]:

definition $\mathit{frob}\ P\ Q \equiv \exists x\ y.\ P\ x \wedge Q\ y$

theorem *frobI* [*intro*]: $P\ x \Longrightarrow Q\ y \Longrightarrow \mathit{frob}\ P\ Q$

unfolding *frob-def* **by** *blast*

theorem *frobE* [*elim*]: $\mathit{frob}\ P\ Q \Longrightarrow (\bigwedge x\ y.\ P\ x \Longrightarrow Q\ y \Longrightarrow C) \Longrightarrow C$

unfolding *frob-def* **by** *blast*

Here the *blast* method refers to generic Classical Reasoning facilities of Isabelle [15] which may be instantiated to object-logics that provide a rule for classical contradiction. Any realistic object-logic would certainly incorporate such proof tools.

Nevertheless we argue that automated reasoning does not address the above issue adequately. First, there are still three variants of essentially the same logical specification (definition / introduction / elimination). Second, unbounded proof search tends to introduce a factor of uncertainty (it may fail unexpectedly for bigger applications).

Going beyond predicate logic, we propose to use inductive definitions to express the original idea of *frob P Q* more directly. The set-theoretic version of inductive definitions in Isabelle/HOL [13] has been recently refined by Stefan Berghofer to work directly with predicates. So we define *frob P Q* as a proposition depending on parameters:

inductive *frob* **for** $P\ Q$ **where** $P\ x \Longrightarrow Q\ y \Longrightarrow \mathit{frob}\ P\ Q$

Isabelle/HOL turns this specification into a primitive definition and recovers the introduction rule and its inversion (the elimination rule) as theorems. Since these are declared as [*intro*] and [*elim*], we may reason with *frob P Q* immediately as seen before. The **inductive** element uses deterministic proof construction inside, which means it scales up to larger applications, avoiding the dangers of full automated reasoning.

Inductive definitions are far more powerful than illustrated here. More advanced applications involve recursive occurrences of the inductive property, and induction becomes the primary proof principle. Inductive proofs of structured statements raise some additional questions that are addressed by the *induct* method of Isabelle/Isar [21].

5 Conclusion

Isabelle/Isar has been successfully used in much larger applications than the examples presented here. The general concepts for producing theories bit-by-bit are essentially the same, starting from primitive notions and continuing towards complex formalizations. So the common style of Isar proof texts has been represented adequately here.

More elaborate applications will require advanced proof methods (like the generic Simplifier and Classical Reasoner included in Isabelle, specific procedures for fragments of arithmetic etc.), and further derived specification mechanism (like **inductive** shown above). The Isar framework is sufficiently flexible to incorporate such additional components built around the Pure framework.

Apart from gaining this flexibility, the impact of Pure on Isar goes even further: the very idea of composing Isar sub-proofs stems from rule refinement in Pure. In retrospection, structured natural deduction appears to have been present in Pure all the time, we only had make it accessible to the user. Thus the original idea of Isabelle [11, 12] to challenge the predominance of classical first-order logic is continued by Isabelle/Isar.

This "puristic" approach to reasoning, without auxiliary logical connectives getting in between, can be observed in the **obtain / obtains** element in its extreme. Here the user works directly with collections of local parameters and premises, without having to introduce and eliminate auxiliary propositions involving $\exists$, $\wedge$, $\vee$ first.

On the other hand, *if* such conglomerates of logical connectives need to be accommodated by single steps, Isar will require extra verbosity in repeating some intermediate statements. In this respect, Isar is "declarative" in an extreme sense, where Mizar proofs would admit immediate transformations of the pending problem. Consequently, Mizar usually proceeds quicker in decomposing statements of predicate logic, but Isar does not require any such decomposition, if problems are presented in Pure form. Apart from being theoretically pleasing, the Pure approach also turns out as a genuine practical advantage, e.g. in structured induction proofs involving compound statements [21].

So Isar proofs emphasize statements from the user application, which are composed directly according to natural deduction principles. The need of predicate logic as an auxiliary device is significantly reduced, replacing it by primary proof elements. This is an important insight gained from the experience with Isabelle/Isar in the past few years.

References

[1] C. Ballarin. Locales and locale expressions in Isabelle/Isar. In S. Berardi et al., editors, *Types for Proofs and Programs (TYPES 2003)*, LNCS 3085, 2004.

[2] G. Bauer and M. Wenzel. Calculational reasoning revisited — an Isabelle/Isar experience. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2001*, LNCS 2152, 2001.

[3] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 1940.

[4] G. Gentzen. Untersuchungen über das logische Schließen. *Math. Zeitschrift*, 1935.

[5] J. Harrison. A Mizar mode for HOL. In J. Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: TPHOLs '96*, LNCS 2152, 1996.

[6] F. Kammüller, M. Wenzel, and L. C. Paulson. Locales: A sectioning concept for Isabelle. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics (TPHOLs '99)*, LNCS 1690, 1999.

[7] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4), 1991.

[8] T. Nipkow. Structured proofs in Isar/HOL. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs (TYPES 2002)*, LNCS 2646, 2003.

[9] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. LNCS 2283. 2002.

[10] L. C. Paulson. Natural deduction as higher-order resolution. *Journal of Logic Programming*, 3, 1986.

[11] L. C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.

[12] L. C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*. Academic Press, 1990.

[13] L. C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In A. Bundy, editor, *Automated Deduction (CADE-12)*, LNAI 814, 1994.

[14] L. C. Paulson. Generic automatic proof tools. In R. Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*. MIT Press, 1997.

[15] L. C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3), 1999.

[16] P. Rudnicki. An overview of the MIZAR project. In *1992 Workshop on Types for Proofs and Programs*. Chalmers University of Technology, Bastad, 1992.

[17] P. Schroeder-Heister. A natural extension of natural deduction. *Journal of Symbolic Logic*, 49(4), 1984.

[18] A. Trybulec. Some features of the Mizar language. Presented at a workshop in Turin, 1993.

[19] 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'99*, LNCS 1690, 1999.

[20] M. Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. PhD thesis, Institut für Informatik, TU München, 2002.

[21] M. Wenzel. Structured induction proofs in Isabelle/Isar. In J. Borwein and W. Farmer, editors, *Mathematical Knowledge Management (MKM 2006)*, LNAI 4108, 2006.

[22] M. Wenzel and L. C. Paulson. Isabelle/Isar. In F. Wiedijk, editor, *The Seventeen Provers of the World*, LNAI 3600. 2006.

[23] F. Wiedijk. Mizar: An impression. Unpublished, 1999.

[24] F. Wiedijk. Mizar light for HOL light. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2001*, LNCS 2152, 2001.

[25] F. Wiedijk. Writing a Mizar article in nine easy steps. Unpublished, 2006.

[26] F. Wiedijk and M. Wenzel. A comparison of the mathematical proof languages Mizar and Isar. *Journal of Automated Reasoning*, 29(3-4), 2002.