Constructing Isabelle Proofs in a Proof Refinement Calculus

Christophe Depasse

Université catholique de Louvain,
Dept. of Computing Science and Engineering,
Place Sainte-Barbe 2, B-1348 Louvain-la-Neuve, Belgium
depasse@info.ucl.ac.be

Abstract. We present a framework to develop structured proofs in Isabelle. The Isabelle proofs are expressed in an algebraic calculus for proof composition and refinement. We benefit from the power of Isabelle and from the algebraic structure of proofs. The Isabelle meta-logic is translated in the algebraic calculus, translating theorems into abstract proofs. Deduction rules are expressed as refinement rules; the premises are replaced by the structured proof of the conclusion. These refinement rules may be used as easily as the corresponding Isabelle rules, knowing the premises to be used. This approach allows us to integrate other theorem provers in the calculus, providing a general framework for various theorem provers.

1 Introduction

For a number of years, many theorem proving assistants have been designed [1]. They are used for theorem proving and system verification. Some of them are well established in the community, e.g. Isabelle [2], PVS [3], Coq [4]. Recently, more attention is being paid to the user interfaces and in particular, to the presentation of the proofs.

In [5], Merriam and Harrison review critical characteristics of an efficient interface. So far, we don't know any system that can meet all the requirements stated. However powerful, the theorem proving assistants conceal useful information from the user, in the sense that the proof itself is hidden, focusing on proving formulae. The proof is embedded in the process of verification. As a consequence, major problems arise concerning proof planning and reflection, i.e. the ability to read, understand and develop the proof.

To tackle these proof presentation problems, we use a proof refinement calculus [6] to represent and construct proofs. This algebraic calculus uses abstraction and refinement as essential techniques for proofs. This approach greatly enhances understanding of the deep structure of proofs. It is well known that proofs are akin to programs, and that programs are best understood by using algebra and refinement [7].

The calculus is designed as a generic framework for proofs. To obtain a practical system for developing structured proofs, we can integrate a theorem proving

assistant in the calculus. Then, we benefit from the theorem prover power, and from the proof structure and refinement of the calculus.

The main contribution of this paper is to integrate the Isabelle system in the calculus. Moreover, we show how this integration enhances the construction of understandable proofs. The construction of proof trees in the Isabelle system has already been studied [8], but never outside the Isabelle system. Other systems like PVS and Coq have also been studied. We chose the Isabelle system for clarity. Indeed, the Isabelle meta-logic and the proof refinement calculus can be integrated smoothly, and the main deduction rules are simple enough to have straightforward corresponding refinement rules. As far as we know, proof abstraction and refinement have not been used before to enhance an existing system. Previous works on proof presentation rely on the translation of proofs into various languages, e.g. natural language or mathematical vernacular. But we believe that in these other approaches, the lack of understandability is replaced by syntax overflow and/or ambiguity, because the abstraction level of proofs is not improved.

The paper is organized as follows. First, we informally recall the proof refinement calculus, built on the concepts of abstract proofs, refinement order and composition of proofs. In Section 3, we describe how the Isabelle meta-logic can be translated into the proof refinement calculus. In Section 4, we show how refinement rules, corresponding to Isabelle rules, arise naturally in the proof refinement calculus. In Section 5, we give a brief example of refinement. Finally, Section 6 contains our conclusions.

2 The Proof Refinement Calculus

This paper is based on a proof refinement calculus [6, 9]. This calculus considers proofs as objects at various levels of abstraction. Intuitively, a proof is an argument of a formula, the level of details required relies on the confidence in the argument. With this view, a formula is as a very abstract proof of itself : if you agree that the formula is valid, then it is, if not, you must refine the proof to a more detailed proof. An effective proof of the formula is also an object in the calculus, but a concrete one with sufficient details. One can establish an order between proofs, from very abstract to very concrete ones, and proof sketches in between :

... $\sqsupseteq$ more abstract proof $\sqsupseteq$ .. $\sqsupseteq$ proof sketch $\sqsupseteq$ .. $\sqsupseteq$ more concrete proof $\sqsupseteq$ ... .

This refinement order may be read : (i) from left to right, from abstract to concrete proofs, i.e. $\sqsupseteq$-refinement amounts to backward refinement, (ii) from right to left, from concrete to abstract proofs, i.e. reversed $\sqsupseteq$-refinement amounts to forward refinement. The reverse $\sqsupseteq$-refinement is denoted by $\sqsubseteq$, with the property $P \sqsubseteq Q$ iff $Q \sqsupseteq P$. "Forward (backward) refinement" is abbreviated to "refinement" when the context is clear.

The ordering of proofs yields a complete lattice. The meet $A \bigsqcap B$ is the most abstract proof refining (i.e. more concrete than) A and B. The join $A \bigsqcup B$ is

the most concrete proof for which A and B are more concrete proofs. Moreover, proofs are built using sequential composition, expressed by the $\fatsemi$ operator. Thus $A \fatsemi B$ is the proof A followed by the proof B. The sequential composition is not commutative. Indeed, the development of proofs relies on successive arguments which make little sense without the ordering. We also have the $\mapsto$ and $\mapsfrom$ operators which represent the entailment of proofs. Intuitively, $A \mapsto B$ says that whenever we have A, then we also have B. Or in other words, the proof B is developed assuming the proof A.

Also, the concept of abstract proofs induces the notion of validity of proofs. A proof is said to be valid if it is an argument that can be verified as "correct". For example, $A \mapsto A$ is valid from the structure of the proof. Thus, a proof may be : (i) an effective proof, that is a valid proof, and (ii) an attempt of proof, that is a proof term not yet validated, which may be "correct" or "incorrect". The proof refinement calculus supports the verification of proof validity, but not of proof invalidity. In general, the more concrete a proof is, the easier it is to verify its validity.

To be able to decide whether a proof is valid, a proof object $\mathbf{1}$ is introduced. It represents validity, and is the identity for the $\fatsemi$ operator. Moreover, we want to be more liberal when verifying validity. For example, $(B \fatsemi A) \mapsto A$ should be valid since A is in the antecedents. But it can not be refined to $\mathbf{1}$ within the algebra. Therefore, for the purpose of verifying the validity, we introduce three additional axioms preserving validity :

$$\begin{array}{rcll} A \fatsemi B & \sqsupseteq & B \fatsemi A & \text{(commutativity)} \\ A \fatsemi A & \sqsupseteq & A & \text{(idempotence)} \\ A & \sqsupseteq & A \fatsemi B & \text{(weakening)} \ . \end{array}$$

A proof refinement using these axioms is denoted by $\sqsupseteq_{val}$. A proof P is valid iff it can be refined by $\mathbf{1}$ modulo the additional axioms, i.e. $P \sqsupseteq_{val} \mathbf{1}$. In the previous example, $(B \fatsemi A) \mapsto A$ can be refined to $\mathbf{1}$ by using the weakening and commutativity axioms. However, keep in mind that the sole purpose of this enriched algebra is the verification of validity; these axioms are undesirable when developing proofs. With these axioms you loose the fundamental structure of the proof. Thus, a typical process for proving a formula is : first, refine it to a complete proof, then verify the validity of this proof.

$$\text{formula} \sqsupseteq \ldots \sqsupseteq \text{concrete proof} \sqsupseteq_{val} \ldots \sqsupseteq_{val} \mathbf{1} \ .$$

Finally, two properties are required on the operators :

$$\begin{array}{rcl} A \mapsto B \ \sqsupseteq C & \equiv & B \sqsupseteq \ A \fatsemi C \\ B \mapsfrom A \ \sqsupseteq C & \equiv & B \sqsupseteq \ C \fatsemi A \ . \end{array}$$

Intuitively, if C is a proof that whenever A is valid, B is valid, then, to prove B, we only have to prove A and C. Conversely, if we only have to prove A and C to have B, then when we want B to be valid knowing A is valid, it suffices to prove that C is valid. These properties highlight the Galois connections [10] between

operators. Since sequential composition is not commutative, the operators $\mapsto$ and $\mapsfrom$ differ; this allows us to distinguish between entailing and being entailed.

With these definitions and requirements, except the validity axioms, we obtain a rich structure of quantale [11], which is in fact equivalent to a standard Kleene algebra [12].

Terminology. From now on, we abbreviate "proof refinement calculus" into "proof calculus".

3 The Isabelle Meta-Logic in the Proof Calculus

In this section, we show how to translate the meta-logic of Isabelle in the syntax of the proof calculus. The main idea is to translate the proving part of the Isabelle system in the proof calculus. This way, we can use the proof calculus instead, manipulating structured proofs instead of meta-level formulae. First, we present the principle of this translation. Then, we detail the translation of each Isabelle meta-level operator.

3.1 Principle of the Mapping

We construct a mapping from meta-level formulae, which deals with truth values, to proof objects, dealing with validity values :

$$\tau : \text{Isabelle meta-level formulae} \rightarrow \text{proof objects} .$$

Every meta-level formula Φ is translated into $\tau(\Phi)$ which is valid if and only if Φ is true. For example, the induction principle for naturals, using a schematic variable $?n$,

$$[\![P(0)\,;\, \wedge x.P(x) \Longrightarrow P(Suc(x))]\!] \Longrightarrow P(?n)$$

is translated into the proof term

$$(\bigsqcap x : \tau(P(x)) \mapsto \tau(P(suc(x)))) \fatsemi \tau(P(0)) \mapsto \bigsqcap ?n : \tau(P(?n)) \ .$$

The atomic proof objects, valid or not, represent the "atomic" meta-level formulae, true or false, i.e. the meta-level formulae which do not contain any meta-level connective. In the translation, each meta-level connective is mapped into the operator of the proof calculus having the same semantics modulo the validity axioms. In the example, $\Longrightarrow$ is translated into $\mapsto$ and $\wedge$ into $\bigsqcap$.

In the next subsections, we will detail the following translation sketches :

$$\begin{array}{lcl} \tau(\Phi \Longrightarrow \Psi) & \approx & \tau(\Phi) \mapsto \tau(\Psi) \\ \tau(\wedge x.\Phi) & \approx & \bigsqcap x : \tau(\Phi) \\ \tau(A \equiv B) & \approx & \bigsqcap X : \tau(X(A)) \doteq \tau(X(B)) \\ \tau(\Phi(?a)) & \approx & \bigsqcap ?a : \tau(\Phi) \ , \end{array}$$

where Φ and Ψ are Isabelle meta-level formulae, i.e. of the type *prop*, A and B are Isabelle formulae of any type. These sketches have to be made precise because various type constraints and syntactical restrictions remain implicit, for the sake of simplicity.

3.2 The $\Longrightarrow$ Operator

The implication $\Phi \Longrightarrow \Psi$ is true if and only if Ψ is true whenever Φ is true. Hence, the proof object corresponding to $\Phi \Longrightarrow \Psi$ must be valid if and only if $\tau(\Psi)$ is valid whenever $\tau(\Phi)$ is valid. This proof object is $\tau(\Phi) \mapsto \tau(\Psi)$.

Remark. The Isabelle formula $\Phi_1 \Longrightarrow \Phi_2 \Longrightarrow \Phi$ (A) is thus translated into $\tau(\Phi_1) \mapsto \tau(\Phi_2) \mapsto \tau(\Phi)$ (B). In Isabelle, the formula (A) is *abbreviated* by $[\![\Phi_1; \Phi_2]\!] \Longrightarrow \Phi$. In the proof calculus, the object (B) is *equal* to $\tau(\Phi_2) \fatsemi \tau(\Phi_1) \mapsto \tau(\Phi)$, by the *Galois connection*. In short, the Isabelle formula $[\![\Phi_1; \Phi_2]\!] \Longrightarrow \Phi$ is translated into $\tau(\Phi_2) \fatsemi \tau(\Phi_1) \mapsto \tau(\Phi)$. Note that the translation is based on the validity properties of the proof objects. That means that the validity axioms may be used. For example, $\tau(\Phi_2) \fatsemi \tau(\Phi_1)$ in general differs from $\tau(\Phi_1) \fatsemi \tau(\Phi_2)$ since $\fatsemi$ is not commutative. But from the validity point of view, these are equal since the commutativity axiom is available.

3.3 The $\wedge$ Operator

Consider the Isabelle formula $\wedge x.P(x)$. This formula is true if $P(x)$ is true for an arbitrary parameter x. The corresponding proof object is simply $\bigsqcap x : \tau(P(x))$. Indeed, by definition of $\bigsqcap$, this proof object is valid if and only if $\tau(P(x))$ is valid whatever the parameter x, thus $\wedge x.P(x)$ is true if and only if $\bigsqcap x : \tau(P(x))$ is valid.

3.4 The $\equiv$ Operator

The $\equiv$ operator is used for definitions. This operator is particular: its arguments may be of any type, provided they are of the same type, but its result is a meta-level truth value. For example, in $m + n \equiv rec(m, n, \lambda x\, y.Suc(y))$, the arguments $m + n$, $rec(...)$ are of type "nat", and the whole formula is of type "prop", a meta-level truth value. Thus, each such expression must be translated in the proof refinement calculus.

To translate $\equiv$-formulae into atomic proof objects would be incorrect. Indeed, the $\equiv$-formula $\Phi \equiv (\Psi_1 \Longrightarrow \Psi_2)$ would be translated into an atomic proof object which still contains a $\Longrightarrow$-operator that is not translated. This must be avoided. Instead, we would like the translation of $\Phi \equiv (\Psi_1 \Longrightarrow \Psi_2)$ to be something like $\tau(\Phi) \doteq \tau(\Psi_1 \Longrightarrow \Psi_2)$. The $\doteq$-operator does not exist in the proof calculus, but can be defined as an abbreviation for a meet of entailments :

$$\tau(\Phi) \doteq \tau(\Psi) \quad \triangleq \quad \tau(\Phi) \mapsto \tau(\Psi) \bigsqcap \tau(\Psi) \mapsto \tau(\Phi) \ .$$

Of course, this can be applied only where Φ and Ψ are meta-level formulae. In the case of arguments A and B that are formulae of any type, we translate $A \equiv B$ in such a way that the corresponding proof object allows us to replace A by B anywhere and conversely while developing proofs. The corresponding proof object must consider any possible context. This leads to the definition

$$\tau(A \equiv B) \triangleq \bigsqcap X : (X : [\alpha] \Longrightarrow \text{prop}), X\text{atomic} : \tau(X(A)) \doteq \tau(X(B)) \ ,$$

where $[\alpha] \Longrightarrow \mathit{prop}$ is a subterm in the proof calculus denoting the restriction on the type of X in the Isabelle type system; and "X atomic" means that X does not contain any meta-level connective.

3.5 Schematic Variables

Above, we have defined the translation of a part of the meta-logic of Isabelle in the proof calculus; this part is the one that does not use schematic variables. Here, we tackle meta-level formulae containing schematic variables such as $?y$. The idea is to "quantify" the formulae over the schematic variables. For instance,

$$?y \Longrightarrow Q \ \text{ is translated into } \ \bigsqcap ?y : (\tau(?y) \mapsto \tau(Q)) \ .$$

The resulting proof object is valid iff $\tau(?y) \mapsto \tau(Q)$ is valid for every instantiation of $?y$. Note that this translation allows us to instantiate the schematic variables as in the Isabelle system. In the previous example, $?y$ may be instantiated with $\wedge x.R(x)$:

$$\bigsqcap ?y : (\tau(?y) \mapsto \tau(Q)) \ \sqsubseteq \ (\bigsqcap x : \tau(R(x))) \mapsto \tau(Q) \ .$$

Every formula of Isabelle is firstly translated using the previously defined translations and then it is "quantified" with $\bigsqcap$.

Notation. Consider a proof object $\tau(\Phi)$ and let SV be the syntactic sequence of schematic variables occurring in Φ. We denote the meet over schematic variables by $\bigsqcap SV : \tau(\Phi)$.

3.6 Properties of the Operators

The meta-logic of Isabelle also contains logical rules that are mostly available in the proof calculus. In the proof calculus, these rules are inherited from the algebraic structure. For example, the counterpart of the rule

$$\frac{\Phi \mapsto \Psi \quad \Phi}{\Psi} \quad \text{is the refinement} \quad \tau(\Psi) \sqsupseteq \tau(\Phi) \fatsemi (\tau(\Phi) \mapsto \tau(\Psi)) \ .$$

However, the rules working on lambda terms must be declared as valid when working on Isabelle terms. For example, in the case of the abstraction, we must declare the proof object $\bigsqcap x, a, b : a \doteq b \ \mapsto \ (\lambda x.a) \doteq (\lambda x.b)$ valid.

4 The Isabelle Rules in the Proof Calculus

In this section we review the main deduction rules of Isabelle. In Isabelle, each rule application is a proof component by itself : the proof consists in the sequence of rules applied. In the proof calculus, successive applications of refinement rules are not considered as proofs, but rather as steps to obtain proofs. We have two orthogonal views of proofs : (i) the proof terms which are the logical justifications of formulae, and highlight the structure of these justifications; (ii) the refinement steps which serve to contruct proof terms; these refinement steps correspond to Isabelle proofs. As a consequence, an inference rule

$$\frac{A}{B}$$

corresponds to a proof refinement $P_A \sqsubseteq B$, where P_A includes all proof steps entailing B from A. We insist : A represents the *formula* from which B can be deduced, whereas P_A represents the *proof* whose validity justifies the validity of B.

We first describe how to translate the Isabelle rules into refinement rules which allow to construct the proofs of the formulae deduced. We begin with the most important rule, the resolution rule. The lifting rules are presented next.

4.1 Principle of the Mapping

Let us consider an Isabelle rule such as

$$\frac{A \quad B}{C} \ .$$

In the proof calculus, we should obtain a rule such as

$$\frac{\tau(A) \sqsupseteq_{val} \mathbf{1} \qquad \tau(B) \sqsupseteq_{val} \mathbf{1}}{\tau(C) \sqsupseteq_{val} \mathbf{1}} \ .$$

This means that we should be able to show that whenever $\tau(A)$ and $\tau(B)$ are valid, $\tau(C)$ is valid. Thus, to be consistent with the spirit of proof refinement, we must refine $\tau(C)$ to a proof $P_{\tau(A),\tau(B)}$ which is valid whenever $\tau(A)$ and $\tau(B)$ are valid. This leads to a backward proof refinement of the form

$$\tau(C) \sqsupseteq P_{\tau(A),\tau(B)} \quad \text{i.e.} \quad P_{\tau(A),\tau(B)} \sqsubseteq \tau(C) \ .$$

To simplify the presentation, we consider formulae of the form $[\![\Phi_1; \Phi_2; \Phi_3]\!] \Longrightarrow \Phi$ instead of the more general $[\![\Phi_1; ...; \Phi_n]\!] \Longrightarrow \Phi$. The results can easily be generalized. Also, the mapping τ is left implicit where it is clear that it must be applied.

4.2 Resolution as Proof Refinement

We show how to find a refinement rule to construct proofs similarly to resolution in Isabelle. We apply the principle given in the previous section.

Basic Idea. We first illustrate the method on a simpler version of the resolution rule, without schematic variables :

$$\frac{A \Longrightarrow B \quad B \Longrightarrow C}{A \Longrightarrow C} \ .$$

In the proof calculus we should have :

$$\frac{\tau(A) \mapsto \tau(B) \ \sqsupseteq_{val} \mathbf{1} \qquad \tau(B) \mapsto \tau(C) \ \sqsupseteq_{val} \mathbf{1}}{\tau(A) \mapsto \tau(C) \ \sqsupseteq_{val} \mathbf{1}} \ .$$

This inference rule is of course correct, but it does not construct the proof of the conclusion. We must refine the conclusion so as to obtain a concrete proof of the latter :

$$\begin{array}{l} \tau(A) \mapsto \tau(C) \\ \sqsupseteq \\ (\tau(B) \leftarrow\!\shortmid (\tau(A) \mapsto \tau(B))) \mapsto \tau(C) \ . \end{array}$$

Note that we have refined the conclusion, which amounts to backward proving. However, the resolution rule is forward. The forward refinement, corresponding to the Isabelle-like in forward proving, is expressed as

$$\begin{array}{l} (\tau(B) \leftarrow\!\shortmid (\tau(A) \mapsto \tau(B))) \mapsto \tau(C) \\ \sqsubseteq \\ \tau(A) \mapsto \tau(C) \ , \end{array}$$

where $P \sqsubseteq Q$ iff $Q \sqsupseteq P$. This forward refinement $\sqsubseteq$ goes from concrete proofs to abstract ones : the refinement $\sqsupseteq$ is then read from right to left.

Forward Proofs. Let us now use the basic idea for the Isabelle resolution rule. Consider for example

$$\frac{\begin{array}{c} [\![\Psi_1; \Psi_2; \Psi_3]\!] \Longrightarrow \Psi \\ [\![\Phi_1; \Phi_2; \Phi_3]\!] \Longrightarrow \Phi \end{array}}{([\![\Phi_1; \Psi_1; \Psi_2; \Psi_3; \Phi_3]\!] \Longrightarrow \Phi)\sigma} \ (\Psi\sigma \equiv \Phi_2\sigma) \ , \tag{1}$$

where $E\sigma$ denotes E to which the substitution σ has been applied on schematic variables. Rule (1) is written as follows in the proof calculus :

$$\frac{\begin{array}{c} (\bigsqcap SV_1 : (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \mapsto \Psi)) \sqsupseteq \mathbf{1} \\ (\bigsqcap SV_2 : (\Phi_3 \fatsemi \Phi_2 \fatsemi \Phi_1 \mapsto \Phi)) \sqsupseteq \mathbf{1} \end{array}}{(\bigsqcap SV_1 \oplus SV_2 : (\Phi_3 \fatsemi \Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \fatsemi \Phi_1 \mapsto \Phi))\sigma \sqsupseteq \mathbf{1}} \ (\Psi\sigma \equiv \Phi_2\sigma) \ . \tag{2}$$

In (2), SV_1 and SV_2 are the syntactic sequences of schematic variables occurring in any Φ_i or Ψ_i, respectively, and $SV_1 \oplus SV_2$ is the concatenation of SV_1 and SV_2. We assume no schematic variable with the same identifiers occurs in both SV_1 and SV_2; if needed, we rename schematic variables before applying the rule. Recall the difference between (i) the verification that a proof is valid, or in other

words, that a formula is correct, and (ii) the construction of a concrete proof from which the validity can be verified. In the same way, we first verify the correctness of (2), and then we use this verification to construct the concrete proof of the conclusion.

We verify the correctness of (2) using the proof calculus as follows :

$$
\begin{array}{l}
(\bigsqcap SV_1 \oplus SV_2 : (\Phi_3 \fatsemi (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1) \fatsemi \Phi_1 \mapsto \Phi))\sigma \\
\sqsupseteq_{val} \{\text{validity check using first assumption}\} \\
(\bigsqcap SV_1 \oplus SV_2 : (\Phi_3 \fatsemi \Psi \fatsemi \Phi_1 \mapsto \Phi))\sigma \\
\sqsupseteq \{\Psi\sigma \equiv \Phi_2\sigma\} \\
(\bigsqcap SV_1 \oplus SV_2 : (\Phi_3 \fatsemi \Phi_2 \fatsemi \Phi_1 \mapsto \Phi))\sigma \\
\sqsupseteq \{\text{property of meet}\} \\
(\bigsqcap SV_2 : \Phi_3 \fatsemi \Phi_2 \fatsemi \Phi_1 \mapsto \Phi) \\
\sqsupseteq_{val} \{\text{validity check using second assumption}\} \\
\mathbf{1} \ .
\end{array}
$$

In this refinement, we have proved that the conclusion of (2) is valid whenever the premises are. This allows to perform the proof in the same way as done with Isabelle. But our goal is to construct a proof term. That is, to start with the premises to be used, and then to construct an explicit proof whose validity entails the validity of the conclusion of (2). In the refinement hereabove, we refine the conclusion to the second assumption. Why is this not the right proof ? Because we have used the first premise merely as a refinement rule. Instead, it should explicitly appear in the constructed proof. Thus, in order to construct proofs effectively, we have to postpone the use of the validity of the premises to the step where the proof validity is checked :

$$
\begin{array}{l}
(\bigsqcap SV_1 \oplus SV_2 : (\Phi_3 \fatsemi (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1) \fatsemi \Phi_1 \mapsto \Phi))\sigma \\
\sqsupseteq \{\text{property of meet, antimonotonicity}\} \\
(\bigsqcap SV_1 \oplus SV_2 : \\
\ (\Phi_3 \fatsemi (\Psi \leftarrowtail (\bigsqcap SV_1 : (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \mapsto \Psi))) \fatsemi \Phi_1 \mapsto \Phi))\sigma \\
\sqsupseteq \{\Psi\sigma \equiv \Phi_2\sigma\} \\
(\bigsqcap SV_1 \oplus SV_2 : \\
\ (\Phi_3 \fatsemi (\Phi_2 \leftarrowtail (\bigsqcap SV_1 : (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \mapsto \Psi))) \fatsemi \Phi_1 \mapsto \Phi))\sigma \\
\sqsupseteq \{\text{property of meet, redundant use of SV1}\} \\
\bigsqcap SV_2 : \Phi_3 \fatsemi (\Phi_2 \leftarrowtail (\bigsqcap SV_1 : (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \mapsto \Psi))) \fatsemi \Phi_1 \mapsto \Phi \\
\sqsupseteq_{val} \{\text{validity check using both assumptions}\} \\
\mathbf{1} \ .
\end{array}
$$

Here we have a proof of the conclusion, which integrates all relevant details, and the validity of which can easily be checked from the validity of the premises. In this proof term, it is required that under the three premises Φ_3, $\Phi_2 \leftarrowtail (\bigsqcap SV_1 ...)$, and Φ_1, the consequent Φ is valid. This requirement to establish the validity of the conclusion of rule (2) is more general than the resolution principle as stated initially. Indeed, the premise $\Phi_2 \leftarrowtail (\bigsqcap SV_1 ...)$ does not require that Φ_2 unifies with Ψ, but only that Φ_2 itself has $(\bigsqcap SV_1 ...)$ as a more concrete proof.

To sum up, the resolution rule (1) is expressed in the proof calculus by the following forward refinement rule, using the reverse symbol $\sqsubseteq$:

$$\begin{array}{l} \bigsqcap SV_2 : \Phi_3 \fatsemi (\Phi_2 \leftarrowtail (\bigsqcap SV_1 : (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \mapsto \Psi))) \fatsemi \Phi_1 \mapsto \Phi \\ \sqsubseteq \{\Psi\sigma \equiv \Phi_2\sigma\} \\ (\bigsqcap SV_1 \oplus SV_2\sigma : (\Phi_3 \fatsemi \Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \fatsemi \Phi_1 \mapsto \Phi))\sigma \ . \end{array} \tag{3}$$

This rule can be instantiated automatically given the premises to be used. So, the proof calculus can be used very much in the same way as Isabelle, but in addition a proof of the conclusion is constructed, and the user can see a *structured justification* of his/her concluded formula.

Backward Proofs. Actually, although the resolution rule is forward, it is also used in Isabelle to *simulate* backward proving. In a similar way, the forward refinement rule (3) can be modified to simulate backward proving. Indeed, we may work with refinements from abstract to concrete proofs, that is backward proving, and then retrieve the corresponding Isabelle rules to obtain the same result. We only need to introduce an auxiliary condition.

A "backward" proof in Isabelle starts with a proof state $\Phi \Longrightarrow \Phi$ and then, using resolution steps, changes it to $[\![\Phi_1; ...; \Phi_n]\!] \Longrightarrow \Phi$. This is repeated until all Φ_i are axioms. As a consequence, the proof of Φ should appear in the conclusion. Thus, we have to move the proof to the other side of the refinement. Thanks to the Galois connection, $A \leftarrowtail B \sqsupseteq C$ is equivalent to $A \sqsupseteq C \fatsemi B$. We can transform (3) into the equivalent forward refinement :

$$\begin{array}{l} (\bigsqcap SV_2 : \Phi_3 \fatsemi \Phi_2 \fatsemi \Phi_1 \mapsto \Phi) \\ \sqsubseteq \{\Psi\sigma \equiv \Phi_2\sigma\} \\ (\bigsqcap SV_1 \oplus SV_2 : \\ \ (\Phi_3 \fatsemi (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1) \fatsemi (\bigsqcap SV_1 : (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \mapsto \Psi)) \fatsemi \Phi_1 \mapsto \Phi))\sigma \ . \end{array} \tag{4}$$

This forward refinement, as in Isabelle, *simulates* backward proving. Indeed, we develop proofs using forward refinements from more concrete to more abstract proofs, and then retrieve the backward proof in the antecedents of the conclusion. Also, if Φ contains schematic variables which are instantiated, we have not a proof of Φ but a proof of the instantiated Φ.

Fortunately, under a simple assumption, we can work on the antecedents only, by backward proving, and then retrieve the corresponding forward refinements on the form (4). The additional condition is that Φ does not contain any schematic variable. Then, the forward refinement (4) can be transformed into

$$\begin{array}{l} (\bigsqcup SV_2 : \Phi_3 \fatsemi \Phi_2 \fatsemi \Phi_1) \mapsto \Phi \\ \sqsubseteq \{\Psi\sigma \equiv \Phi_2\sigma\} \\ (\bigsqcup SV_1 \oplus SV_2 : \\ \ (\Phi_3 \fatsemi (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1) \fatsemi (\bigsqcap SV_1 : (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \mapsto \Psi)) \fatsemi \Phi_1))\sigma \mapsto \Phi \ . \end{array}$$

Note that the refinement bears on the antecedents only. Indeed, in the proof calculus

$$\begin{array}{lll} A & & B \mapsto C \\ \sqsupseteq & \text{implies} & \sqsupseteq \\ B & & A \mapsto C \end{array} \qquad \text{i.e.} \qquad \begin{array}{l} A \mapsto C \\ \sqsubseteq \\ B \mapsto C \ . \end{array}$$

Thus, we may leave the conclusion Φ and work on the antecedents only :

$$\begin{array}{l} (\bigsqcup SV_2 : \Phi_3 \fatsemi \Phi_2 \fatsemi \Phi_1) \\ \sqsupseteq \{\Psi\sigma \equiv \Phi_2\sigma\} \\ (\bigsqcup SV_1 \oplus SV_2 : \\ \ (\Phi_3 \fatsemi \Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \fatsemi (\bigsqcap SV_1 : (\Psi_3 \fatsemi \Psi_2 \fatsemi \Psi_1 \mapsto \Psi)) \fatsemi \Phi_1))\sigma \ . \end{array} \tag{5}$$

This $\sqsupseteq$-refinement goes from abstract to concrete proofs, i.e. we develop backward proofs. Moreover, from a refinement step using (5), we can retrieve the corresponding Isabelle states.

4.3 Lifting as Proof Refinement

Other important rules in the Isabelle system are the lifting rules. They are mainly used to lift the rules before the use of resolution. We only give the translation of these lifting rules for breviety. The justification of the translation can be found in [13].

Lifting over Assumptions. In Isabelle, the lifting over assumptions Θ can be stated as

$$\frac{[\![\Phi_2; \Phi_1]\!] \Longrightarrow \Phi}{[\![\Theta \Longrightarrow \Phi_1; \Theta \Longrightarrow \Phi_2]\!] \Longrightarrow (\Theta \Longrightarrow \Phi)} \ . \tag{6}$$

The corresponding forward refinement is

$$\begin{array}{l} \bigsqcap SV : (\Theta \fatsemi (\Theta \mapsto \Phi_2 \fatsemi \Theta \mapsto \Phi_1)) \mapsto \Phi_2 \fatsemi \Phi_1 \fatsemi (\Phi_2 \fatsemi \Phi_1 \mapsto \Phi) \\ \sqsubseteq \\ \bigsqcap SV : (\Theta \mapsto \Phi_2 \fatsemi \Theta \mapsto \Phi_1) \mapsto (\Theta \mapsto \Phi) \ . \end{array} \tag{7}$$

Indeed, to prove the second proof object, we show that knowing θ and θ entails Φ_i, we have Φ_1, Φ_2 and these Φ_i entails Φ. Of course, this is valid if $\Phi_2 \fatsemi \Phi_1 \mapsto \Phi$.

Lifting over Parameters. The Isabelle rule for lifting over parameters is

$$\frac{[\![\Phi_2 \fatsemi \Phi_1]\!] \Longrightarrow \Phi}{[\![\wedge x.{\Phi_2}^x; \wedge x.{\Phi_1}^x]\!] \Longrightarrow \wedge x.\Phi^x} \ , \tag{8}$$

where Φ^x stands for Φ where all schematic variables $?a$ have been transformed to $?a'(x)$, $?a'$ being a new schematic function-variable. This allows to express the dependence on x in the terms prefixed with $\wedge x$. The corresponding forward refinement is straightforward :

$$\begin{array}{l} \bigsqcap SV : \Phi_1 \fatsemi \Phi_2 \mapsto \Phi \\ \sqsubseteq \\ (\bigsqcap SV^x : ((\bigsqcap x : {\Phi_1}^x) \fatsemi (\bigsqcap x : {\Phi_2}^x)) \mapsto (\bigsqcap x : \Phi^x)) \ , \end{array} \tag{9}$$

where SV^x is the syntactic sequence of schematic variables resulting from the mapping $?a \mapsto ?a'$ on the variables in SV.

5 Example

To illustrate the proof calculus applied to the Isabelle system, we present the refinement of a common theorem, which is the same as in [8]. This highlights that proof refinement enhances understandability.

The proof objects are objects written in the context of the HOL object logic. We will use the following part of HOL :

Connectives. The connectives used are :

$$\begin{array}{ll} Tr & :: bool \Longrightarrow prop \quad \forall :: ([\alpha] \Longrightarrow bool) \Longrightarrow bool \\ \longrightarrow & :: bool \Longrightarrow bool \Longrightarrow bool \quad \exists :: ([\alpha] \Longrightarrow bool) \Longrightarrow bool \ . \end{array}$$

Note that Tr is the coercion from the type *bool* to the meta-level truth value *prop*. This induce that every formula $Tr\ P$ is translated by $\tau(Tr\ P)$. Below, the application of Tr is left understood.

Inference Rules. The inference rules are presented in the order of their use, and are already expressed in the proof calculus language :

$$\begin{array}{ll} impI & \triangleq \bigsqcap ?P, ?Q : (\tau(?P) \mapsto \tau(?Q)) \mapsto \tau(?P \longrightarrow ?Q) \\ allI & \triangleq \bigsqcap ?P : (\bigsqcap x : \tau(?P\ x)) \mapsto \tau(\forall x.\ ?P\ x) \\ exE & \triangleq \bigsqcap ?P, ?Q : (\bigsqcap x : (\tau(?P\ x) \mapsto \tau(?Q))) \fatsemi \tau(\exists x.\ ?P\ x) \mapsto \tau(?Q) \\ exI & \triangleq \bigsqcap ?P, ?x : \tau(?P\ ?x) \mapsto \tau(\exists x.\ ?P\ x) \\ allE & \triangleq \bigsqcap ?P, ?x, ?R : (\tau(?P\ ?x) \mapsto \tau(?R)) \fatsemi \tau(\forall x.\ ?P\ x) \mapsto \tau(?R) \end{array}$$

The notation $lifted\_over(r, \overline{pa})$ represents the rule r lifted over the parameters and/or assumptions $\overline{pa}$.

Theorem. $(\exists x.\ \forall y.\ P\ x\ y) \longrightarrow (\forall y.\ \exists x.\ P\ x\ y)$

Proof Refinement. We could refine the theorem by applying the same rules to the same objects as in the Isabelle proof. However, we use the same rules in the same order, but not on the same objects, as we aim at using the proof calculus to construct a more structured proof. The objects refined at each step are underlined.

$$\begin{array}{ll} & \underline{\tau((\exists x.\ \forall y.\ P\ x\ y) \longrightarrow (\forall y.\ \exists x.\ P\ x\ y))} \\ \sqsupseteq & \{impI\} \\ & (\tau(\exists x.\ \forall y.\ P\ x\ y) \mapsto \underline{\tau(\forall y.\ \exists x.\ P\ x\ y))} \\ & \fatsemi impI \\ \sqsupseteq & \{allI\} \\ & (\tau(\exists x. \forall y.\ P\ x\ y) \\ & \ \mapsto \underline{(\bigsqcap y : \tau(\exists x.\ P\ x\ y))} \fatsemi allI) \\ & \fatsemi impI \end{array}$$

$$
\begin{array}{l}
\sqsupseteq \{exE\} \\
\quad (\tau(\exists\, x.\ \forall\, y.\ P\ x\ y) \\
\quad\ \mapsto (\underline{\textstyle\bigsqcap x : (\tau(\forall\, y.\ P\ x\ y) \mapsto (\bigsqcap y : \tau(\exists\, x.\ P\ x\ y)))}) \\
\quad\quad \fatsemi \tau(\exists\, x.\ \forall\, y.\ P\ x\ y) \\
\quad\quad \fatsemi exE \fatsemi allI) \\
\quad \fatsemi impI \\
\sqsupseteq \{exI \text{ lifted over } y,\ \tau(\forall\, y.\ P\ x\ y) \text{ and } x\} \\
\quad (\tau(\exists\, x.\ \forall\, y.\ P\ x\ y) \\
\quad\ \mapsto (\underline{\textstyle\bigsqcup ?x : \bigsqcap x : (\tau(\forall\, y.\ P\ x\ y) \mapsto (\bigsqcap y : \tau(P\ (?x\ x\ y)\ y)))}) \\
\quad\quad \fatsemi lifted\_over(exI, y, \tau(\forall\, y.\ P\ x\ y), x) \\
\quad\quad \fatsemi \tau(\exists\, x.\ \forall\, y.\ P\ x\ y) \fatsemi exE \fatsemi allI) \\
\quad \fatsemi impI \\
\sqsupseteq \{allE \text{ lifted over } y,\ \tau(\forall\, y.\ P\ x\ y) \text{ and } x\} \\
\quad (\tau(\exists\, x.\ \forall\, y.\ P\ x\ y) \\
\quad\ \mapsto (\underline{\textstyle\bigsqcup ?x, ?x1 : \bigsqcap x : (\tau(\forall\, y.\ P\ x\ y) \mapsto (\bigsqcap y : \tau(P\ x\ (?x1\ x\ y)) \mapsto \tau(P\ (?x\ x\ y)\ y)))}) \\
\quad\quad \fatsemi (\bigsqcap x : (\tau(\forall\, y.\ P\ x\ y) \mapsto \tau(\forall\, xa.\ P\ x\ xa))) \\
\quad\quad \fatsemi lifted\_over(allE, y, \tau(\forall\, y.\ P\ x\ y), x) \\
\quad\quad \fatsemi lifted\_over(exI, y, \tau(\forall\, y.\ P\ x\ y), x) \fatsemi \tau(\exists\, x.\ \forall\, y.\ P\ x\ y) \fatsemi exE \fatsemi allI) \\
\quad \fatsemi impI \\
\sqsupseteq \{\text{instantiation of } ?x \text{ and } ?x1\} \\
\quad (\tau(\exists\, x.\ \forall\, y.\ P\ x\ y) \\
\quad\ \mapsto (\bigsqcap x : (\tau(\forall\, y.\ P\ x\ y) \mapsto (\bigsqcap y : \tau(P\ x\ y) \mapsto \tau(P\ x\ y)))) \\
\quad\quad \fatsemi (\bigsqcap x : (\tau(\forall\, y.\ P\ x\ y) \mapsto \tau(\forall\, xa.\ P\ x\ xa))) \\
\quad\quad \fatsemi lifted\_over(allE, y, \tau(\forall\, y.\ P\ x\ y), x) \\
\quad\quad \fatsemi lifted\_over(exI, y, \tau(\forall\, y.\ P\ x\ y), x) \fatsemi \tau(\exists\, x.\ \forall\, y.\ P\ x\ y) \fatsemi exE \fatsemi allI) \\
\quad \fatsemi impI
\end{array}
$$

This is the final proof. It can be refined by 1, modulo the validity axioms. Thus, the theorem is proved. Compared to proof trees [8], this proof is simpler to read and understand. However, it contains only the main ideas of the proof, and not the details such as instantiations, whilst proof trees do. Here, we must use the refinement steps to deduce the right instantiations.

6 Conclusion

We have shown how to integrate the Isabelle system into the proof refinement calculus. We mapped the formulae into the same semantic domain as proofs. By adapting the Isabelle meta-language, we have extended the system with the concept of abstract proofs. This interpretation of the Isabelle meta-logic allows us to use the proof calculus as a deductive system, by using refinements corresponding to Isabelle deduction rules. Our main aim has been to convince the reader that this approach allows us to work with the same ease as in Isabelle, while developing structured proofs. Thanks to the use of abstraction as fundamental basis of the calculus, we may explore proofs at different levels of abstraction. For

example, proofs may be presented hierarchically. However, these proofs can be quite big, and it is not clear whether these proofs and refinement steps can be easily and efficiently compressed.

Although we focused on Isabelle, our approach could be applied to other existing systems. Indeed, the proof refinement calculus is generic enough to provide a common basis for various theorem proving assistants. Moreover, this would allow various systems to interoperate, on the restriction that they use an appropriate version of a common theory. However, the integration of an other theorem prover might not be as smooth as the one presented here. For example, to integrate PVS, we can translate only the goals in the proof calculus, *every* rule is translated into an object of the calculus.

Acknowledgments

The author would like to thank Michel Sintzoff for the helpful discussions on this research, and Raphaël Collet, Micha Janssens and Yves Kamp for their comments on this paper.

References

1. *Database of existing mechanized reasoning systems*, Stanford Univ., 1999, http://www-formal.stanford.edu/clt/ARS/systems.html .
2. L. C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828, Springer, 1994.
3. S. Owre, N. Shankar, J. M. Rushby, D. W. J. Stringer-Calvert. *PVS Prover Guide*. Computer Science Lab., SRI International, 1999.
4. B. Barras et al. *The Coq Proof Assistant Reference Manual - Version 6.2*. INRIA, Rocquencourt, 1998.
5. N. A. Merriam, M. D. Harrisson. Evaluating the Interfaces of Three Theorem Proving Assistants. In : F. Bodart and J. Vanderdonckt (eds.), *Proc. 3rd International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems*, Springer, 1996.
6. M. Simons, M. Sintzoff. Algebraic Composition and Refinement of Proofs. In : M. Johnson (ed.), *Algebraic Methodology and Software Techhnology*, pages 494-508. LNCS 1349, Springer, 1997.
7. R. Bird, O. de Moor. *Algebra of Programming*, Prentice Hall, 1997.
8. s. Berghofer, T. Nipkow. Proof Terms for Simply Typed Higher Order Logic. In : M. Aagaard, J. Harrison (eds.), *Theorem Proving in Higher Order Logics*, pages 38-52. LNCS 1869, Springer, 2000.
9. M. Simons. *The Presentation of Formal Proofs*. R. Oldenbourg Verlag, 1997.
10. H. Herrlich, M. Hušek. Galois Connections. In : A. Melton (ed.), *Mathematical Foundations of Programming Semantics*, pages 122-134. LNCS 239, Springer, 1985.
11. K. I. Rosenthal. *Quantales and their Applications*. Longman Scientific & Technical, 1990.
12. J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.
13. C. Depasse. *Constructing Isabelle Proofs in a Proof Refinement Calculus*. Research Report, UCL, 2001, ftp://ftp.info.ucl.ac.be/pub/reports/2001/rr2001-02.ps.gz .