Calculational reasoning revisited

an Isabelle/Isar experience

Gertrud Bauer and Markus Wenzel

Technische Universität München
Institut für Informatik, Arcisstrasse 21, 80290 München, Germany
http://www.in.tum.de/~bauerg/
http://www.in.tum.de/~wenzelm/

Abstract. We discuss the general concept of calculational reasoning within Isabelle/Isar, which provides a framework for high-level natural deduction proofs that may be written in a human-readable fashion. Setting out from a few basic logical concepts of the underlying meta-logical framework of Isabelle, such as higher-order unification and resolution, calculational commands are added to the basic Isar proof language in a flexible and non-intrusive manner. Thus calculational proof style may be combined with the remaining natural deduction proof language in a liberal manner, resulting in many useful proof patterns. A case-study on formalizing Computational Tree Logic (CTL) in simply-typed set-theory demonstrates common calculational idioms in practice.

1 Introduction

A proof by calculational reasoning basically proceeds by forming a chain of intermediate results that are meant to be composed by basic principles, such as transitivity of $=/</\leq$ (or similar relations). More advanced calculations may even involve substitution, which in the case of inequalities usually includes monotonicity constraints. In informal mathematics, this kind of proof technique is routinely used in a very casual manner. Whenever mathematicians write down sequences of mixed equalities or inequalities, underline subexpressions to be replaced etc. then it is likely that they are doing calculational reasoning.

In fact, calculational reasoning has been occasionally proposed as simple means to rephrase mathematical proof into a slightly more formal setting (e.g. [2, 1]), which does not necessarily include machine-checking of proofs, of course. Observing that logical equivalence and implication may be just as well used in calculations, some have even set out to do away with traditional natural-deduction style reasoning altogether [5], although that discipline does not appeal to everyone.

Nevertheless, calculational reasoning offers a relatively simple conceptual basis to build tools for logical manipulations. The popular Math$\int$pad tool supports manipulation of algebraic expressions in a systematic way; it has recently even acquired means for formal proof checking [18], using PVS as the backend.

The Mizar system [17, 12, 22] focuses on formal proof in common mathematics style in the first place. It also offers a mechanism for iterated equality reasoning, which shall serve here as an example for calculations within a formal setting. The following trivial proof is taken from article #185 of the Mizar library [11].

```
theorem Th1:
  for X,Y being set holds union {X,Y,{}} = union {X,Y}
proof
   let X,Y be set;
   thus union {X,Y,{}} = union ({X,Y} U {{}}) by ENUMSET1:43
                .= union {X,Y} U union {{}} by ZFMISC_1:96
                .= union {X,Y} U {} by ZFMISC_1:31
                .= union {X,Y};
end;
```

In Mizar "`thus`" indicates that the subsequent statement is meant to solve a pending goal. The continued equality sign "`.=`" indicates that the actual result shall emerge from a number of individual equations, each being proven separately and the results composed by transitivity behind the scenes.

The present paper discusses quite general concepts of calculational reasoning that may be expressed within the Isabelle/Isar framework for human-readable proof documents [19, 20]. Isar provides a high-level view on natural deduction, but is open to incorporate additional derived language elements such as those for calculational reasoning. Thus techniques of natural deduction and calculations may be conveniently used together in the same proof, enabling the writer to apply the most appropriate one for the particular situation at hand.

So the two paradigms may coexist peacefully, and even benefit from each other. There need not be a conflict of natural deduction versus calculational reasoning, as is occasionally raised by followers of Dijkstra's proof format [5].

Before going into further details, we shall see how the above example works out within the Isar proof language.[1] First of all, we observe that it could be easily finished by a single stroke of an automated proof method of Isabelle [13].

theorem $\bigcup\{X,\ Y,\ \{\}\} = \bigcup\{X,\ Y\}$ **by** *auto*

In fact, many calculations in the Mizar library are rather trivial from the perspective of automated proof tools available in Isabelle, HOL, PVS etc., which indicates that Mizar's builtin automation does not handle equality too well. On the other hand, we would usually be less lucky with automated tools, once that the applications get more "realistic". In contrast, well-defined concepts of structured proof (such as calculations) provide means to arrange formal reasoning in a robust and scalable manner, being oriented towards the human reader (and writer) of proof texts, rather than the machine. Automated methods would then find there proper place in solving local proof obligations only.

The subsequent version mimics the original Mizar proof as closely as possible.

[1] All formal proofs given in this paper have been processed with Isabelle99-2.

theorem $\bigcup\{X,\ Y,\ \{\}\} = \bigcup\{X,\ Y\}$
proof −
  have $\bigcup\{X,\ Y,\ \{\}\} = \bigcup(\{X,\ Y\} \cup \{\{\}\})$ **by** *auto*
  also have $\ldots = \bigcup\{X,\ Y\} \cup \bigcup\{\{\}\}$ **by** *auto*
  also have $\ldots = \bigcup\{X,\ Y\} \cup \{\}$ **by** *auto*
  also have $\ldots = \bigcup\{X,\ Y\}$ **by** *auto*
  finally show $\bigcup\{X,\ Y,\ \{\}\} = \bigcup\{X,\ Y\}$ **.**
qed

Isar provides an explicit mechanism to finish a calculation (unlike Mizar). In the canonical style of writing calculations this is used to reiterate the final result, sparing readers to determine it themselves. Calculations are not restricted to a fixed scheme, but may be freely composed via a few additional commands (**also** and **finally** encountered above) and the "..." notation for the right-hand side of the most recent statement (see §2.4). Thus the above text merely turns out as an idiomatic expression within a more general language framework (see §3).

We now inspect a bit further how the proof actually proceeds, and recall that the original Mizar proof basically imitates a simplification process. The justifications of intermediate claims (as indicated by **by**) are only of marginal interest here. We look more closely at the transformational process of the equations involved, which is represented at the top level as a plain transitive chain, but essentially performs a few substitution steps. Since Isabelle/Isar handles substitution as well, we may explain these technical details directly within the formal text.

theorem $\bigcup\{X,\ Y,\ \{\}\} = \bigcup\{X,\ Y\}$
proof −
  have $\{X,\ Y,\ \{\}\} = \{X,\ Y\} \cup \{\{\}\}$ **by** *auto*
  also have $\bigcup(\{X,\ Y\} \cup \{\{\}\}) = \bigcup\{X,\ Y\} \cup \bigcup\{\{\}\}$ **by** *auto*
  also have $\bigcup\{\{\}\} = \{\}$ **by** *auto*
  also have $\bigcup\{X,\ Y\} \cup \{\} = \bigcup\{X,\ Y\}$ **by** *auto*
  finally show $\bigcup\{X,\ Y,\ \{\}\} = \bigcup\{X,\ Y\}$ **.**
qed

Apparently, the result of a calculation need not be the first left-hand side being equal to the last right-hand side, as more general rules get involved.

2 Foundations of calculational reasoning

2.1 Logical preliminaries

We use standard mathematical notation as far as possible; just note that we write lists as $[x_1,\ \ldots,\ x_n]$ and $\overline{a}\ @\ \overline{b}$ for appending lists $\overline{a}$ and $\overline{b}$.

Our basic logical foundations are that of the Isabelle/Pure framework [13], a minimal higher-order logic with universal quantification $\bigwedge x.\ P\ x$ and implication $A \Longrightarrow B$; the underlying term language is that of simply-typed λ-calculus, with application $f\ x$ and abstraction $\lambda x.\ f$. Examples are presented in the object-logic

Isabelle/HOL [13], which extends Pure by common connectives ($\neg$, $\longrightarrow$, $\wedge$, $\vee$, $\forall$, $\exists$ etc.), the classical axiom, and Hilbert's choice operator.

By *theorem* we denote the set of derivable theorems of Pure, we write $\vdash \varphi$ to indicate that proposition φ is a theorem; furthermore let *facts* be the set of lists over *theorem*. Theorems of Pure actually represent (derived) rules of the embedded object logic. The main (derived) operations of Pure are *resolution* (back-chaining, generalized modus ponens) and proof by *assumption* [13] — both are quite powerful primitives as they may involve higher-order unification. We write $r \cdot \overline{a}$ for the resulting theorem of resolving facts $\overline{a}$ in parallel into rule r. A *goal* is represented as a theorem, which is $\vdash \varphi \Longrightarrow \varphi$ initially and gets refined by resolution to become $\vdash \varphi$ eventually [13]; note that Isabelle/Isar is already content with a goal state that is finished up to proof-by-assumption.

2.2 The Isabelle/Isar proof language

The Isar proof language provides a general framework for human-readable natural deduction proofs [19, 20]; its basic concepts are somewhat oriented towards the basic Isabelle/Pure framework, which happens to offer a good basis for primitive operations of natural deduction (especially resolution $r \cdot \overline{a}$).

The Isar language consists of 12 primitives [20, Appendix A]: "**fix** $x :: \tau$" and "**assume** a: A" augment the context, **then** indicates forward chaining, "**have** a: A" and "**show** a: A" claim local statements (the latter also solves some pending goal afterwards), "**proof** m" performs an initial proof step by applying some method, "**qed** m" concludes a (sub-)proof, **{ }** and **next** manage block structure, "**note** $a = \overline{b}$" binds reconsidered facts, and "**let** $p = t$" abbreviates terms via higher-order matching (the form "(**is** p)" may be appended to any statement).

Basic proof methods are: "$-$" to do nothing, *this* to resolve facts directly (performs $\mathit{goal} \cdot \mathit{this}$), and "(*rule* r)" to apply a rule resolved with facts (performs $\mathit{goal} \cdot (r \cdot \mathit{this})$). Arbitrary automated proof tools may be used as well, such as *simp* for Isabelle's Simplifier, and *auto* for a combination of several tools [13].

Standard abbreviations include *?thesis* for the initial claim (head of the proof), and "..." for the right-hand side of the latest (finished) statement; *this* refers the result from the previous step. Default methods are *rule* for **proof**, and "$-$" for **qed**. Further derived proof commands are "**by** m_1 m_2" for "**proof** m_1 **qed** m_2", "**..**" for "**by** *rule*", "**.**" for "**by** *this*", and "**from** $\overline{a}$" for "**note** $\overline{a}$ **then**".

Isar's natural deduction kernel directly corresponds to the underlying logical framework. A meta-level statement may be established as follows.

have $\bigwedge x\ y\ z.\ A \Longrightarrow B \Longrightarrow C$
proof $-$
  fix $x\ y\ z$ **assume** A **and** B
  show C ⟨*proof*⟩
qed

In reality, such rule statements would typically emerge from a different claim being refined by an initial proof method, used instead of "$-$" encountered here.

2.3 Calculational sequences

From a syntactical point of view, the essence of a calculational proof is what we shall call a *calculational sequence*: let *calculation* be freely generated by the constructors *start*: *facts* $\to$ *calculation* and *continue*: *calculation* $\to$ *facts* $\to$ *calculation*. Apparently, any *calculation* simply represents a non-empty list of facts. We fine-tune our notation and write canonical calculational sequences *continue* $(\ldots (\mathit{continue}\ (\mathit{start}\ a_1)\ a_2) \ldots\ a_n)$ concisely as $a_1 \circ a_2 \cdots \circ a_n$.

An *interpreted calculation sequence* shall be any *result* achieved by mapping *start* and *continue* in a primitive recursive fashion. We only consider interpretations of *calculation* back into *facts*, i.e. *result*: *calculation* $\to$ *facts*; we also fix *result* $(\mathit{start}\ a) = a$. There is only one degree of freedom left to specify *result* $(c \circ a)$ in order to give an interpretation of *continue* steps. The following two kinds of calculational steps will be considered within the Isabelle/Isar framework.

(rule-step): specify *result* $(c \circ a) = r \cdot (\mathit{result}\ c\ @\ a)$ where r is a suitable rule taken from a given set $\mathcal{T}$ of *transitivity rules*. We produce a (single) result by applying a rule to the current calculational result plus some new facts.

(accumulation-step): specify *result* $(c \circ a) = \mathit{result}\ c\ @\ a$. We simply collect further facts without applying any rule yet.

As a basic example of interpreted calculation sequences, just fix the singleton set $\mathcal{T} = \{\vdash x = y \Longrightarrow y = z \Longrightarrow x = z\}$ of transitivity rules and only perform rule steps; then we have *result* $(\vdash x_1 = x_2 \circ \vdash x_2 = x_3 \circ \vdash x_3 = x_4) = \vdash x_1 = x_4$. Thus we may represent canonical chains of equations composed by plain transitivity. Alternatively, only perform accumulation steps to achieve *result* $(\vdash \varphi_1 \circ \vdash \varphi_2 \circ \vdash \varphi_2) = [\vdash \varphi_1, \vdash \varphi_2, \vdash \varphi_2]$, i.e. simply get a number of facts collected as a single list. As we shall see later on, even the latter case of seemingly degenerate calculational sequences turns out to be quite useful in practice.

2.4 Calculational elements within the proof language

In the next stage we investigate how suitable proof development systems may provide a language interface for the user to compose calculational sequences.

At first sight, the way taken by Mizar [17] seems to be the obvious one: simply invent concrete syntax for the primitive operations of composing calculational sequences, and make the implementation support this directly — probably with some link to the basic mechanisms of stating and proving facts. This way of "making a system do something particular" is usually limited to just the specific feature one had in mind when planning the implementation — no more, no less.

In Isabelle/Isar we do not hardwire calculational reasoning, but figure out how the process of composing calculational sequences may be mapped into the natural flow of reasoning within the existing Isar framework in a non-intrusive fashion. By adding only a few abbreviations and conventions, we achieve a very general framework for calculational reasoning with minimal effort. The resulting space of possible combined proof patterns shall be explored later on.

First of all, we fix a special facts register called "*calculation*" to hold the current state of the (partially interpreted) sequence the user is currently working on. The start of a calculation shall be determined implicitly, as indicated by *calculation* being empty. Whenever a calculation is finished (by an explicit command to be given below), *calculation* will be reset to await the next sequence to start. The result of a finished sequence is exhibited to the subsequent goal statement as explicitly highlighted facts (their actual use in the subsequent proof is not controlled by the calculational process anymore).

We also wish to exploit Isar's inherent block structure to support nested calculations. So any update operation on *calculation* needs to track the current nesting level, in order to commence a new sequence whenever blocks are opened.

The derived Isar proof commands to maintain the *calculation* register are defined as follows, leaving the policies of initializing and resetting the state implicit.

$$\begin{aligned}
\textbf{also} &\equiv \textbf{note}\ \mathit{calculation} = \mathit{this} && \text{(initial case)}\\
\textbf{also} &\equiv \textbf{note}\ \mathit{calculation} = r \cdot (\mathit{calculation}\ @\ \mathit{this}) && \text{(for some } r \in \mathcal{T})\\
\textbf{finally} &\equiv \textbf{also from}\ \mathit{calculation}\\
\textbf{moreover} &\equiv \textbf{note}\ \mathit{calculation} = \mathit{calculation}\ @\ \mathit{this}\\
\textbf{ultimately} &\equiv \textbf{moreover from}\ \mathit{calculation}
\end{aligned}$$

Here the two main elements are **also** and **moreover**, corresponding to the "rule-steps" and "accumulation-steps" introduced before. The variants of **finally** and **ultimately** finish the current sequence after performing a final step. Due to the forward chaining involved in the **from** operation, the next command has to be a goal statement like **have** or **show** (cf. the Isar semantics given in [19]).

With one more element we arrive at a viable calculational proof language within the Isar framework: the standard term binding "..." refers to right-hand side of the most recent explicit fact statement. This enables the user to include relevant parts of the previous statement in a succinct manner. The "Mizar mode for HOL" [8] provides a similar element, while Mizar [12] uses the "`.=`" construct.

We may now write the previous examples of calculational sequences as follows.

have $x_1 = x_2$ $\langle \mathit{proof} \rangle$
also have $\ldots = x_3$ $\langle \mathit{proof} \rangle$
also have $\ldots = x_4$ $\langle \mathit{proof} \rangle$
finally have $x_1 = x_4$ **.**

In the next calculation we use the ultimate list of accumulated facts to prove a result by a certain rule $r = \vdash \varphi_1 \Longrightarrow \varphi_2 \Longrightarrow \varphi_3 \Longrightarrow \varphi_4$.

have φ_1 $\langle \mathit{proof} \rangle$
moreover have φ_2 $\langle \mathit{proof} \rangle$
moreover have φ_3 $\langle \mathit{proof} \rangle$
ultimately have φ_4 **by** (*rule* r)

Certainly, we may rephrase these calculations as basic natural deduction in backwards style, while performing exactly the same inferences internally.

have $x_1 = x_4$
proof (*rule trans*)
  show $x_1 = x_3$
  proof (*rule trans*)
    show $x_1 = x_2$ ⟨*proof*⟩
    show $x_2 = x_3$ ⟨*proof*⟩
  qed
  show $x_3 = x_4$ ⟨*proof*⟩
qed

have φ_4
proof (*rule r*)
  show φ_1 ⟨*proof*⟩
  show φ_2 ⟨*proof*⟩
  show φ_3 ⟨*proof*⟩
qed

2.5 Rules and proof search

The philosophy of Isar is to keep automated proof tools out of the basic mechanisms of interpreting the high-level structure proof texts. Only linear search over a limited number of choices plus (higher-order) unification is permitted here.

Reconsidering the commands for outlining calculational sequences in Isar (§2.4), we see that there is a single non-deterministic parameter: the rule $r \in \mathcal{T}$ to be selected by the **also** command. As Isar proof texts are interpreted strictly from left to right [19], the subsequent result $\mathit{calculation} = r \cdot (\mathit{calculation} \mathbin{@} \mathit{this})$ has to be achieved from the present facts alone, with the rule instance r determined by the system appropriately. As long as $\mathcal{T}$ only holds (mixed) transitivities of $=/</\leq$ the result is already uniquely determined, e.g. providing facts $\vdash x \leq y$ and $\vdash y < z$ invariably yields $\vdash x < z$.

Isar uses the following refined strategy to support more general rule selections. Assume a canonical order on the rule context $\mathcal{T}$, and let $a = \mathit{calculation} \mathbin{@} \mathit{this}$ be the input given to the present calculational step. Now enumerate the members r of $\mathcal{T}$, then enumerate the canonical sequences of results $r \cdot a$ as obtained by parallel higher-order unification and back-chaining of a with r. Finally filter this raw result sequence to *disallow mere projections* of a; in other words remove those results b that do not make any actual "progress", in the sense that the conclusion of b is already present in one of the members of the list a.

This strategy subsumes the simple case of unique results considered before, but also does a good job at substitution: let us declare $\vdash P\ x \Longrightarrow x = y \Longrightarrow P\ y$ and $\vdash y = x \Longrightarrow P\ x \Longrightarrow P\ y$ to be tried after the plain transitivities considered so far. The expression $x = y$ only requires plain first-order unification, with a unique most-general result. The critical part is to solve $P\ x$ against the other expression, which is a genuine higher-order problem. Resulting unifiers will assign a certain λ-term to P that abstracts over possible occurrences of sub-expression x. Here the standard strategy [13] is to start with a solution with all occurrences, followed by all possible partial occurrences in a fixed order, and finish with *no* occurrences. Note that the latter case is the only solution if x does not occur at all, which is actually a pathological case for our purpose, as it collapses the substitution rules to $\vdash P \Longrightarrow x = y \Longrightarrow P$ and $\vdash y = x \Longrightarrow P \Longrightarrow P$.

Thus by filtering out mere projections of the original facts, a basic calculational rule-step is able to produce a sensible result, where *all* occurrences of a certain

sub-expression may be replaced by an equal one (cf. the final example given in §1). Replacing only some occurrences does *not* work, though, as there is no way to specify the intended result beforehand. In the latter case, it is better to use plain transitivity together with the Simplifier to justify the next step.

Substitution with inequalities (involving additional monotonicity constraints) works as well, see §3 for common patterns. The notion of "progress" in the filtering strategy needs to ignore local premises to detect degenerate cases properly.

3 Idioms of calculational reasoning

The space of possible calculational expressions within Isar is somewhat open-ended, due to the particular way that calculational primitives have been incorporated into the proof language. Certainly, creative users of Isabelle/Isar may invent further ways of calculational reasoning at any time. Here we point out possible dimensions of variety, and hint at practically useful idiomatic patterns. Our subsequent categories are guided by the way that primitive calculational sequences (cf. §2.3) may be mapped to Isar proof configurations (cf. §2.4).

3.1 Variation of rules

The most basic form is a plain transitive chain of equations, cf. the second Isar example in §1. Mixed transitivities may be used as follows; observe that the canonical ending (with a single-dot proof) exhibits the result explicitly.

have $x_1 \le x_2$ ⟨*proof*⟩
also have $\ldots \le x_3$ ⟨*proof*⟩
also have $\ldots = x_4$ ⟨*proof*⟩
also have $\ldots < x_5$ ⟨*proof*⟩
also have $\ldots = x_6$ ⟨*proof*⟩
finally have $x_1 < x_6$ **.**

Likewise, we may use further combinations of relations such as antisymmetry, as long as there is a clear functional mapping from facts to the result, and no serious conflict with other rules.

have $x \le y$ ⟨*proof*⟩ **also have** $y \le x$ ⟨*proof*⟩
finally have $x = y$ **.**

We have already covered substitution of equals by equals near the end of §1 (and §2.5); with inequalities this works out quite similarly.

have $A = B + x + C$ ⟨*proof*⟩
also have $x \le y$ ⟨*proof*⟩
finally — $\vdash (\bigwedge x\ y.\ x \le y \Longrightarrow B + x + C \le B + y + C) \Longrightarrow A \le B + y + C$
have $A \le B + y + C$ **by** *simp*

The rule used here is $\vdash a = f\ b \Longrightarrow b \le c \Longrightarrow (\bigwedge u\ v.\ u \le v \Longrightarrow f\ u \le f\ v) \Longrightarrow a \le f\ c$, which has 3 premises, but we have only filled in two facts during the

calculation; the remaining monotonicity constraint has been left as a hypothesis of the result, which eventually was solved by the final simplification step. The hard part of instantiating the side-condition has already been performed during the calculation, with the relevant propositions given in the text. We see how high-level proof outlining nicely cooperates with dumb automated reasoning.

In very simple cases, one may as well provide all 3 facts in the first place. For example, see the phrase "**moreover note** *AG-mono*" that appears in §4.3.

We may also calculate directly with logical propositions, approaching the original proof style of [5]. The following pattern essentially achieves "light-weight" natural deduction, by implicit use of the *modus ponens* rule.

have $A \longrightarrow B \longrightarrow C$ ⟨*proof*⟩
also have A ⟨*proof*⟩
also have B ⟨*proof*⟩
finally have C **.**

Certainly, transitivity of "$\longrightarrow$" may be used as well. On the other hand, chaining of implications is more conveniently expressed directly by Isar's **then** primitive (cf. §2.2), circumventing the overhead of explicit logical connectives altogether.

3.2 Variation of conclusions

Recall that the actual business of managing the calculational process finishes with the concluding **finally** or **ultimately** command, which just offers the result with forward-chaining indicated (cf. §2.4). The next command may be any kind of goal, such **have**, **show**, or even the powerful **obtain** [3, 20].

Any such claim has to be followed by a proof. The most basic one is "**.**", meaning that the goal statement actually reiterates the calculational result directly. Another useful idiom is to feed the result (which may be just a number accumulated facts) into a single rule (with several premises), e.g. see the proof of *AG-AG* in §4.3. One may even generalize this principle to use arbitrary automated methods, resulting in some kind of "big-step" inferences. Without the calculational infrastructure, the latter mode of operation would usually require a lot of name references for intermediate facts, which tend to degrade readability.

3.3 Variation of facts

In virtually all calculational schemes discussed so far, the facts to be placed into the chain are produced as local statements "**have** φ ⟨*proof*⟩". Nevertheless, any Isar language element that produces facts may be used in calculations. This includes **note** to recall existing theorems, or other goal elements such as **show** or **obtain**, or even context commands such as **assume**. See §4.3 for some uses of "**moreover note**", and §4.4 for "**also note**". Combinations with **obtain** are very useful in typical computer-science applications (e.g. [21]) where results about representations of syntactic entities are incrementally put together.

The use of **assume** within a calculation represents the most basic case of combining calculational reasoning and natural deduction, e.g. within an induction.

theorem $(\sum i < n.\ 2 * i + 1) = n^2$
proof (*induct* n)
  show $(\sum i < 0.\ 2 * i + 1) = 0^2$ **by** *auto*
next
  fix n **have** $(\sum i < Suc\ n.\ 2 * i + 1) = 2 * n + 1 + (\sum i < n.\ 2 * i + 1)$ **by** *auto*
  also assume $(\sum i < n.\ 2 * i + 1) = n^2$
  also have $2 * n + 1 + n^2 = (Suc\ n)^2$ **by** *auto*
  finally show $(\sum i < Suc\ n.\ 2 * i + 1) = (Suc\ n)^2$ **.**
qed

The "**also assume**" line indicates substitution with the induction hypothesis.

3.4 Variation of structure

Calculational sequences are basically linear, but arbitrarily many intermediate steps may be taken until the next fact is produced. This includes further nested calculations, as long as these are arranged on a separate level of block structure. See §4.4 for the very common case of using the implicit block structure induced by local proofs, and §4.3 for explicit blocks indicated by braces.

4 Case-study: some properties of CTL

In order to demonstrate how the idiomatic expressions of calculational reasoning are used in practice, we present a case-study of formalizing basic concepts of Computational Tree Logic (CTL) [10, 9] within the simply-typed set theory of HOL.[2] The proofs are mostly by algebraic reasoning over basic set operations.

4.1 CTL formulae

By using the common technique of "shallow embedding", a CTL formula is identified with the corresponding set of states where it holds. Consequently, CTL operations such as negation, conjunction, disjunction simply become complement, intersection, union of sets. We only require a separate operation for implication, as point-wise inclusion is usually not encountered in plain set-theory.

types $\alpha\ ctl = \alpha\ set$
constdefs
  $imp :: \alpha\ ctl \Rightarrow \alpha\ ctl \Rightarrow \alpha\ ctl$ (**infixr** $\rightarrow$ 75)
  $p \rightarrow q \equiv -\ p \cup q$

The CTL path operators are more interesting; they are based on an arbitrary, but fixed model $\mathcal{M}$, which is simply a transition relation over states α.

[2] See also http://isabelle.in.tum.de/library/HOL/CTL/document.pdf

consts *model* :: $(\alpha \times \alpha)$ *set* $\quad (\mathcal{M})$

The operators EX, EF, EG are taken as primitives, while AX, AF, AG are defined as derived ones. We denote by EX p the set of states with a successor state s' (with respect to the model $\mathcal{M}$), such that $s' \in p$. The expression EF p denotes the set of states, such that there is a path in $\mathcal{M}$, starting from that state, such that there is a state s' on the path with $s' \in p$. The expression EG p is the set of all states s, such that there is a path, starting from s, such that for all states s' on the path, $s' \in p$. It is well known that EF p and EG p may be expressed using least and greatest fixed points [10].

constdefs
$EX :: \alpha\ ctl \Rightarrow \alpha\ ctl$ (EX - [80] 90) EX $p \equiv \{s.\ \exists s'.\ (s, s') \in \mathcal{M} \wedge s' \in p\}$
$EF :: \alpha\ ctl \Rightarrow \alpha\ ctl$ (EF - [80] 90) EF $p \equiv lfp\ (\lambda s.\ p \cup \text{EX}\ s)$
$EG :: \alpha\ ctl \Rightarrow \alpha\ ctl$ (EG - [80] 90) EG $p \equiv gfp\ (\lambda s.\ p \cap \text{EX}\ s)$

AX, AF and AG are now defined dually in terms of EX, EF and EG.

constdefs
$AX :: \alpha\ ctl \Rightarrow \alpha\ ctl$ (AX - [80] 90) AX $p \equiv -\ \text{EX} -\ p$
$AF :: \alpha\ ctl \Rightarrow \alpha\ ctl$ (AF - [80] 90) AF $p \equiv -\ \text{EG} -\ p$
$AG :: \alpha\ ctl \Rightarrow \alpha\ ctl$ (AG - [80] 90) AG $p \equiv -\ \text{EF} -\ p$

4.2 Basic fixed point properties

First of all, we use the de-Morgan property of fixed points

lemma *lfp-gfp*: *lfp* $f = -\ gfp\ (\lambda s\ .\ -\ (f\ (-\ s)))$ ⟨*proof*⟩

in order to give dual fixed point representations of AF p and AG p:

lemma *AF-lfp*: AF $p = lfp\ (\lambda s.\ p \cup \text{AX}\ s)$ **by** (*auto simp add*: *lfp-gfp*)
lemma *AG-gfp*: AG $p = gfp\ (\lambda s.\ p \cap \text{AX}\ s)$ **by** (*auto simp add*: *lfp-gfp*)

From the greatest fixed point definition of AG p, we derive as a consequence of the Knaster-Tarski theorem on the one hand that AG p is a fixed point of the monotonic function $\lambda s.\ p \cap \text{AX}\ s$.

lemma *AG-fp*: AG $p = p \cap \text{AX AG}\ p$
proof −
have *mono* $(\lambda s.\ p \cap \text{AX}\ s)$ ⟨*proof*⟩
then show *?thesis* ⟨*proof*⟩
qed

This fact may be split up into two inequalities (merely using transitivity of $\subseteq$, which is an instance of the overloaded $\leq$ in Isabelle/HOL).

lemma *AG-fp*$_1$: AG $p \subseteq p$
proof −
note *AG-fp* **also have** $p \cap \text{AX AG}\ p \subseteq p$ **by** *auto*
finally show *?thesis* **.**
qed

lemma *AG-fp*$_2$: AG $p \subseteq \text{AX AG}\ p$
proof −

note *AG-fp* **also have** $p \cap \text{AX AG } p \subseteq \text{AX AG } p$ **by** *auto*
finally show *?thesis* **.**
qed

On the other hand, we have from the Knaster-Tarski fixed point theorem that any other post-fixed point of $\lambda s.\ p \cap \text{AX } s$ is smaller than AG p. A post-fixed point is a set of states q such that $q \subseteq p \cap \text{AX } q$. This leads to the following co-induction principle for AG p.

lemma *AG-I*: $q \subseteq p \cap \text{AX } q \Longrightarrow q \subseteq \text{AG } p$
by (*simp only*: *AG-gfp*) (*rule gfp-upperbound*)

4.3 The tree induction principle

With the most basic facts available, we are now able to establish a few more interesting results, leading to the *tree induction* principle for AG (see below). We will use some elementary monotonicity and distributivity rules.

lemma *AX-int*: $\text{AX } (p \cap q) = \text{AX } p \cap \text{AX } q$ ⟨*proof*⟩
lemma *AX-mono*: $p \subseteq q \Longrightarrow \text{AX } p \subseteq \text{AX } q$ ⟨*proof*⟩
lemma *AG-mono*: $p \subseteq q \Longrightarrow \text{AG } p \subseteq \text{AG } q$ ⟨*proof*⟩

If a state is in the set AG p it is also in AX p (we use substitution of $\subseteq$ with monotonicity).

lemma *AG-AX*: $\text{AG } p \subseteq \text{AX } p$
proof −
have $\text{AG } p \subseteq \text{AX AG } p$ **by** (*rule AG-fp*$_2$)
also have $\text{AG } p \subseteq p$ **by** (*rule AG-fp*$_1$) **moreover note** *AX-mono*
finally show *?thesis* **.**
qed

Furthermore we show idempotency of the AG operator. The proof is a good example of how accumulated facts may get used to feed a single rule step.

lemma *AG-AG*: $\text{AG AG } p = \text{AG } p$
proof
show $\text{AG AG } p \subseteq \text{AG } p$ **by** (*rule AG-fp*$_1$)
next
show $\text{AG } p \subseteq \text{AG AG } p$
proof (*rule AG-I*)
have $\text{AG } p \subseteq \text{AG } p$ **..**
moreover have $\text{AG } p \subseteq \text{AX AG } p$ **by** (*rule AG-fp*$_2$)
ultimately show $\text{AG } p \subseteq \text{AG } p \cap \text{AX AG } p$ **..**
qed
qed

We now give an alternative characterization of the AG operator, which describes the AG operator in an "operational" way by tree induction: AG p is the set of all states $s \in p$, such that for all reachable states (starting from that state) holds the following condition: if a state lies in p then also will any successor state.

We use the co-induction principle *AG-I* to establish this in a purely algebraic manner.

```
theorem AG-induct: p ∩ AG (p → AX p) = AG p
proof
  show p ∩ AG (p → AX p) ⊆ AG p  (is ?lhs ⊆ ?rhs)
  proof (rule AG-I)
    show ?lhs ⊆ p ∩ AX ?lhs
    proof
      show ?lhs ⊆ p ..
      show ?lhs ⊆ AX ?lhs
      proof −
        {
          have AG (p → AX p) ⊆ p → AX p by (rule AG-fp1)
          also have p ∩ p → AX p ⊆ AX p ..
          finally have ?lhs ⊆ AX p by auto
        }  — (1)
        moreover
        {
          have p ∩ AG (p → AX p) ⊆ AG (p → AX p) ..
          also have ... ⊆ AX ... by (rule AG-fp2)
          finally have ?lhs ⊆ AX AG (p → AX p) .
        }  — (2)
        ultimately have ?lhs ⊆ AX p ∩ AX AG (p → AX p) ..
        also have ... = AX ?lhs by (simp only: AX-int)
        finally show ?thesis .
      qed
    qed
  qed
next
  show AG p ⊆ p ∩ AG (p → AX p)
  proof
    show AG p ⊆ p by (rule AG-fp1)
    show AG p ⊆ AG (p → AX p)
    proof −
      have AG p = AG AG p by (simp only: AG-AG)
      also have AG p ⊆ AX p by (rule AG-AX) moreover note AG-mono
      also have AX p ⊆ (p → AX p) .. moreover note AG-mono
      finally show ?thesis .
    qed
  qed
qed
```

The middle part of this proof provides an example for nested calculations using explicit blocks: the two contributing results (1) and (2), which are established separately by calculations as well, are ultimately put together.

4.4 An application of tree induction

Further interesting properties of CTL expressions may be demonstrated with the help of tree induction; here we show that AX and AG commute.

theorem *AG-AX-commute*: AG AX p = AX AG p
proof −
  have AG AX p = AX p ∩ AX AG AX p **by** (*rule AG-fp*)
  also have ... = AX (p ∩ AG AX p) **by** (*simp only*: *AX-int*)
  also have p ∩ AG AX p = AG p (**is** *?lhs* = *?rhs*)
  proof — (1)
    have AX p ⊆ p → AX p **..**
    also have p ∩ AG (p → AX p) = AG p **by** (*rule AG-induct*)
    also note *Int-mono AG-mono* — (2)
    ultimately show *?lhs* ⊆ AG p **by** *auto*
  next — (1)
    have AG p ⊆ p **by** (*rule AG-fp*$_1$)
    moreover
    {
      have AG p = AG AG p **by** (*simp only*: *AG-AG*)
      also have AG p ⊆ AX p **by** (*rule AG-AX*)
      also note *AG-mono*
      ultimately have AG p ⊆ AG AX p **.**
    }
    ultimately show AG p ⊆ *?lhs* **..**
  qed — (1)
  finally show *?thesis* **.**
qed

This is an example for nested calculation with implicit block structure (1), as managed automatically by **proof**/**next**/**qed**. Naturally, users would complain if calculations in sub-proofs could affect the general course of reasoning! Also note that (2) indicates a non-trivial use of ⊆ substitution into a monotone context.

4.5 Discussion

Our theory of CTL serves as a nice example of several kinds of calculational reasoning, mainly due to the high-level algebraic view on set operations. Alternatively, one could have worked point-wise with explicit set membership. Then the proofs would certainly have become more cumbersome, with many primitive natural deduction steps to accommodate quantified statements.

There is an interesting story about this example. Incidently, it has once served as an assignment for the Isabelle course given in summer 2000 at TU Munich. After the students had been exposed to Isabelle for a few weeks (only the crude tactical part!), the instructors intended to pose a relatively simple "realistic" application of set theory, which turned out to be much harder than expected.

The reason was that on the one hand, the instructors simply started off by developing the theory interactively in Isabelle/Isar, using its proper proof language basically to "think aloud formally". This was a relatively leisurely experience, as it involves only a number of algebraic manipulation, as we have presented here. On the other hand, the students only knew traditional tactic scripts, with that strong bias towards hairy natural deduction operations performed in backwards style. This posed a real problem to them; some students would even proclaim that the assignment was impossible to finish with their present knowledge.

In retrospect, it is understandable that rephrasing the kind of algebraic reasoning we have seen here into tactic scripts is quite cumbersome, even for the expert.

5 Conclusion and related work

We have seen that calculational reasoning in Isabelle/Isar provides a viable concept for arranging a large variety of algebraic proof techniques in a well-structured manner. While requiring only minimal conservative additions to the basic Isar proof engine, we have been able to achieve a large space of useful patterns of calculational reasoning, including mixed transitivity rules, substitution of equals-by-equals, and even substitution by greater (or equal) sub-expressions. The underlying mechanisms of Isabelle/Isar do not need any advanced proof search, apart from plain (higher-order) unification with a simple filtering scheme.

Interestingly, traditional tactic-based interactive proof systems such as (classic) Isabelle, HOL, Coq, PVS etc. lack support for calculational reasoning altogether. This has been addressed several times in the past. Simons proposes tools to support calculational reasoning within tactical proof scripts [14]. Grundy provides an even more general transformational infrastructure for "window inference" [7]. Harrison's "Mizar mode for HOL" simulates a number of concepts of declarative theorem proving on top of the tactic-based hol-light system [8], including calculational reasoning for mixed transitivity rules.

Concerning the class of theorem proving environments for human-readable proofs, its canonical representative Mizar [17, 12] supports a fixed format for iterative equations, with implicit application of both transitivity and general substitution rules. Syme's DECLARE system for declarative theorem proving [15, 16] does not address calculations at all. Zammit outlines a generalized version of Mizar-style calculations for SPL [23], but observes that these have not been required for the examples at hand, so it has not been implemented.

For users of Isabelle/Isar, calculational reasoning has become a useful tool for everyday applications — not just the typical "mathematical" ones [3], but also (abstract) system verification tasks [21]. Calculations fit indeed very well into the general high-level natural deduction framework of Isar, so we may say that calculational reasoning [5] and natural deduction [6] have been finally reconciled.

References

[1] R. J. Back, J. Grundy, and J. von Wright. Structured calculational proof. *Formal Aspects of Computing*, 9, 1997.

[2] R. J. Back and J. von Wright. Structured derivations: A method for doing high-school mathematics carefully. Technical report, Turku Centre for C.S., 1999.

[3] G. Bauer and M. Wenzel. Computer-assisted mathematics at work — the Hahn-Banach theorem in Isabelle/Isar. In T. Coquand, P. Dybjer, B. Nordström, and J. Smith, editors, *Types for Proofs and Programs: TYPES'99*, LNCS, 2000.

[4] Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors. *Theorem Proving in Higher Order Logics: TPHOLs '99*, LNCS 1690, 1999.

[5] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Texts and monographs in computer science. Springer, 1990.

[6] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 1935.

[7] J. Grundy. Window inference in the HOL system. In M. Archer, J. J. Joyce, K. N. Levitt, and P. J. Windley, editors, *Proceedings of the International Workshop on HOL*. ACM SIGDA, IEEE Computer Society Press, 1991.

[8] 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 1125, 1997.

[9] K. McMillan. Lecture notes on verification of digital and hybrid systems. NATO summer school, http://www-cad.eecs.berkeley.edu/~kenmcmil/tutorial/toc.html.

[10] K. McMillan. *Symbolic Model Checking: an approach to the state explosion problem*. PhD thesis, Carnegie Mellon University, 1992.

[11] Mizar mathematical library. http://www.mizar.org/library/.

[12] M. Muzalewski. *An Outline of PC Mizar*. Fondation of Logic, Mathematics and Informatics — Mizar Users Group, 1993.

[13] L. C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828. 1994.

[14] M. Simons. Proof presentation for Isabelle. In E. L. Gunter and A. Felty, editors, *Theorem Proving in Higher Order Logics: TPHOLs '97*, LNCS 1275, 1997.

[15] D. Syme. DECLARE: A prototype declarative proof system for higher order logic. Technical Report 416, University of Cambridge Computer Laboratory, 1997.

[16] D. Syme. Three tactic theorem proving. In Bertot et al. [4].

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

[18] R. Verhoeven and R. Backhouse. Interfacing program construction and verification. In J. Wing and J. Woodcock, editors, *FM99: The World Congress in Formal Methods*, volume 1708 and 1709 of *LNCS*, 1999.

[19] M. Wenzel. Isar — a generic interpretative approach to readable formal proof documents. In Bertot et al. [4].

[20] M. Wenzel. *The Isabelle/Isar Reference Manual*, 2000. Part of the Isabelle distribution, http://isabelle.in.tum.de/doc/isar-ref.pdf.

[21] M. Wenzel. Some aspects of Unix file-system security. Isabelle/Isar proof document, http://isabelle.in.tum.de/library/HOL/Unix/document.pdf, 2001.

[22] F. Wiedijk. Mizar: An impression. Unpublished paper, 1999. http://www.cs.kun.nl/~freek/mizar/mizarintro.ps.gz.

[23] V. Zammit. On the implementation of an extensible declarative proof language. In Bertot et al. [4].