Now f is continuous (exercise!)

R.D. ARTHAN
Lemma 1 Ltd. & Department of Computer Science, University of Oxford
rda@lemma-one.com

A recurring proof obligation in modern mathematics, ranging from textbook exercises to deep research problems, is to show that a given function is a morphism in some category: in analysis and topology, for example, we frequently need to prove that functions are continuous, while in group theory we are constantly concerned with homomorphisms. This paper describes a generic procedure that automatically discharges routine instances of this kind of proof obligation in an interactive theorem prover. The proof procedure has been implemented and found very useful in a mathematical case studies carried out using the ProofPower system

1. INTRODUCTION

Consider the function $f(x) = e^{2\pi ix}$. If we are interested in algebraic properties of f, we will want to know that it is a homomorphism from the additive group of real numbers to the multiplicative group of non-zero complex numbers, while if we are interested in topological properties, we will want to know that it is continuous with respect to the appropriate topologies. As the title reminds us, in traditional mathematical practice, discharging such proof obligations is often routine and they are left as exercises for the reader. When we are using a mechanized theorem-prover to formalise mathematics, these exercises are often tedious to carry out manually and we can very reasonably expect automatic support to help us find the proofs. This paper discusses a simple but effective generic approach to this kind of problem, viewed as the problem of showing that a given set-theoretic function is a morphism in some category of interest, e.g., the category of groups or the category of topological spaces.

Some work on this kind of automation in specific instances has already been done, particularly in the context of proving continuity and differentiability in formalisations of analysis [Har98, Got00, Fle00, CF02, BLM12] . The goal of this paper is to present a more general approach. We begin with a discussion of formalisations of the concrete categories such as **Grp**, **Top**, **Rng** and $\mathbb{R}-$**Vec** that occur so frequently in mathematical practice. The goal is not to formalise any category theory, but rather to use some basic category-theoretic concepts to help structure our thinking about the formalisation. The categories in question generally have a rather different flavour from the categories that commonly occur in computer science. For example, apart from the category of sets, cartesian closed categories are rather rare in mathematics. We then show, by example, how routine proofs that a given function is a morphism in a concrete category can be broken down into two phases: *(i)* represent the candidate function as a composite built from ground functions by morphism-preserving combinators, then *(ii)* decompose the assertion

This work was supported in part by EPSRC grant number EP/K503769/1.

about the composite into (known) assertions about the ground functions. We then show how this approach admits a straightforward implementation using tools that are available “off-the-shelf” in programmable interactive theorem-proving systems such as the various implementations of HOL. We conclude with a discussion of the application of this approach in some mathematical cases studies carried out using the ProofPower system.

2. REPRESENTING A CONCRETE CATEGORY IN HOL

Recall, for example, from [Mac98], that a *concrete* category is a category in which *(i)* each object X can be identified with an *underlying set* $U(X)$ and *(ii)* the morphisms $X \to Y$ can be identified with some collection of the set-theoretic functions $f : U(X) \to U(Y)$. Concrete categories arise in the very common mathematical scenario in which one deals with sets equipped with some extra structure and functions between the sets that “respect” the structure in some suitable sense. Examples of concrete categories abound in modern mathematics: the following table lists just a few to give a flavour.

Name	Objects	Morphisms
Set	All sets	Arbitrary functions
Grp	Groups	Group homomorphisms
$\mathbb{R}-$**Vec**	Real vector spaces	Linear maps
Top	Topological spaces	Continuous functions

Note that a category whose objects are naturally defined as sets but whose morphisms are not functions between the sets may actually be a concrete category under a suitable definition of the underlying set. For example, **Rel**, the category of sets and relations, is a concrete category if one takes $U(X)$ to be the power set $\mathbb{P}(X)$ and identifies a relation R with the function $A \mapsto \{y \mid \exists x \in A \bullet x\ R\ y\}$. An example of a category that is not equivalent to any concrete category is **Toph**, the homotopy category, which has the same objects as **Top**, but where the morphisms $\mathcal{A} \to \mathcal{B}$ comprise the homotopy equivalence classes of continuous functions from $\mathcal{A}$ to $\mathcal{B}$. The proof that **Toph** is not concrete is due to Freyd [Fre70, Fre04].

The general problem we are concerned with is proving that some given set-theoretic function is a morphism in some concrete category of interest. Our goal is to set up a framework for solving this kind of problem automatically in a useful range of practical situations. We will work in the ProofPower implementation of Mike Gordon’s HOL. See [Gor00] for a survey of HOL and its various implementations and [AJ 1] for more information about the ProofPower implementation. We begin with an extended example showing how the category **Top** has been formalised in the ProofPower mathematical case studies [Art14] and discussing how one might automate proofs of morphismhood in that formalisation.

2.1 Representing **Top**

It will be helpful to have in mind an example formalisation of a particular concrete category: we will sketch the development of the category **Top** in the ProofPower mathematical case studies. An object of **Top** is a topological space, which we represent simply by giving the set of open sets[1]: we say τ is a topology and write $\tau \in \mathit{Topology}$ to mean that τ is a collection of sets of elements of some type $'a$ that is closed under arbitrary unions and binary (and hence arbitrary finite) intersections. Thus *Topology* is the class of families of sets of elements of type $'a$ defined as follows in ProofPower-HOL, using the (postfix) powerset type constructor $\mathbb{P}$ to formalise "class", "family" and "set":

$$\begin{array}{|l}
\mathit{Topology} : {'a}\ \mathbb{P}\ \mathbb{P}\ \mathbb{P} \\
\hline
\quad \mathit{Topology} = \\
\quad \{\tau \\
\quad |\quad (\forall V\bullet\ V \subseteq \tau \Rightarrow \bigcup V \in \tau) \\
\quad \land\quad (\forall A\ B\bullet A \in \tau \land B \in \tau \Rightarrow A \cap B \in \tau)\}
\end{array}$$

Here we use a notation for introducing new HOL constants that ProofPower has borrowed from the Z specification language [Spi92]. In this notation, one gives type constraints for the constant or constants being defined (in this case *Topology*) and then, under the horizontal line, one gives a predicate giving the desired defining property of the constant or constants (in this case an equation giving the value of the constant). This maps onto a call of a primitive definitional principle that requires an existence proof for the constants being introduced. The ProofPower-HOL infrastructure includes a range of procedures for discharging the existence proofs and these will automatically discharge the proof obligations in many cases (including all the ones in this paper).

It is pleasant to call the underlying set of a topology its *space*, which we can define as a union, thus:

$$\begin{array}{|l}
\mathit{Space}_T : {'a}\ \mathbb{P}\ \mathbb{P} \to {'a}\ \mathbb{P} \\
\hline
\forall\tau\bullet\ \mathit{Space}_T\ \tau = \bigcup \tau
\end{array}$$

If τ is a topology, we call its elements *τ-open sets*, so $A \in \tau$ formalises the statement that A is τ-open. The cases studies proceed with other standard definitions, e.g., a set A is defined to be τ-closed if its complement $\mathit{Space}_T\ \tau \setminus A$ is τ-open. In particular, and central to the concerns of the present paper, the notion of continuous function is formalised as follows:

[1]Sets are represented in ProofPower as members of a polymorphic type $'a\,SET$ defined to be in one-to-one correspondence with the type of propositional functions on the type $'a$. The syntax for set comprehension is implemented by the parser and pretty-printer which represent $\{x|t\}$ as $\mathit{SetComp}(\lambda x \bullet t)$ where *SetComp* is the representation function for the type $'a\,SET$. In the present paper, we are working in the context of a library theory which offers $\mathbb{P}$ as an abbreviation for SET.

$$
\begin{array}{|l}
\$Continuous : ('a\ \mathbb{P}\ \mathbb{P} \times 'b\ \mathbb{P}\ \mathbb{P}) \to ('a \to 'b)\ \mathbb{P} \\
\hline
\forall \sigma\ \tau \bullet \quad (\sigma,\ \tau)\ Continuous = \\
\quad \{f \\
\quad | \quad (\forall x \bullet\ x \in Space_T\ \sigma \Rightarrow f\ x \in Space_T\ \tau) \\
\quad \wedge \quad (\forall A \bullet\ A \in \tau \Rightarrow \{x \mid x \in Space_T\ \sigma \wedge f\ x \in A\} \in \sigma)\}
\end{array}
$$

Here *Continuous* is a postfix operator, whose operand comprises a pair of topologies, (σ, τ) and whose value is a set of functions. (The "$" above is required to suppress the postfix status, allowing the name to be used in a declaration.) We say "f is (σ, τ)-continuous" and write $f \in (\sigma,\ \tau)\ Continuous$ if f maps the underlying set of σ to that of τ and if for every τ-open set A, the inverse image of A under the restriction of f to the space of σ is σ-open. Here we are working around the fact that HOL functions are total on their types by using a total function to represent its restriction to the space of σ. This is extremely convenient, since we can re-use existing infrastructure for working with total functions. The price is that one has to be careful when stating results involving equality between functions. For present purposes, the advantages considerably outweigh the disadvantages.

In summary, we have formalised the specific category **Top** by defining: *(i)* a class *Topology* to represent $\mathsf{Obj}_{\mathbf{Top}}$, *(ii)* a function $Space_T$ to represent the underlying set operation $U_{\mathbf{Top}}(\_)$, and *(iii)* a subset $(\sigma,\ \tau)\ Continuous$ of the functions from $U(\sigma)$ to $U(\tau)$ to represent $\mathsf{Mor}_{\mathbf{Top}}(\sigma, \tau)$.

2.2 Proving morphismhood in $(\mathbb{R}; \mathsf{exp}, \mathsf{sin}, \mathsf{cos})_{\mathbf{Top}}$

In the title of this section and some later sections, we adopt the following notational convention: if $\mathcal{C}$ is a category, if $o_1, o_2, \ldots$ is a list of objects of $\mathcal{C}$ or operators on the objects of $\mathcal{C}$ and if $m_1, m_2, \ldots$ is a list of morphisms of $\mathcal{C}$ or operators on the morphisms of $\mathcal{C}$, we write $(o_1, o_2, \ldots; m_1, m_2, \ldots)_{\mathcal{C}}$ for the subcategory of $\mathcal{C}$ generated by the o_i and m_j. So $(\mathbb{R}; \mathsf{exp}, \mathsf{sin}, \mathsf{cos})_{\mathbf{Top}}$ is the subcategory of **Top** comprising the single object $\mathbb{R}$ and all functions obtainable by composing the exponential, sine and cosine functions.

$\mathbb{R}$, the space of real numbers with the usual topology, is formalised by specifying the set O_R of open sets:

$$
\begin{array}{|l}
O_R : \mathbb{R}\ \mathbb{P}\ \mathbb{P} \\
\hline
O_R = \{A \mid \forall t \bullet t \in A \\
\quad \Rightarrow \exists x\ y \bullet t \in OpenInterval\ x\ y \wedge OpenInterval\ x\ y \subseteq A\}
\end{array}
$$

The exponential, sine and cosine functions may be characterized in several different ways: in the ProofPower formalisation, they are characterized by the usual differential equations and the definitions are proved consistent using the power series representations.

It would be considered an easy textbook exercise in basic analysis to prove that the function $f(x) = \mathsf{sin}(\mathsf{cos}(\mathsf{exp}(x)))$ is continuous: one simply observes that sin, cos and exp are continuous and that the composition of two continuous functions is continuous. Let us see how we might automate this kind of proof. Since, for the moment, we are only interested in a single topological space O_R, let us write Cts for $(O_R,\ O_R) Continuous$. We will use the ProofPower syntax $Sin\ x,\ \ Cos\ x,\ \ f\ o\ g$ etc., for the formalised mathematics and the textbook syntax $\mathsf{sin}(x)$, $\mathsf{cos}(x)$, $f \circ g$ etc. in the semi-formal narrative. We assume we have already proved the necessary ground facts:

$$\begin{array}{l} \vdash Exp \in Cts \\ \vdash Sin \in Cts \\ \vdash Cos \in Cts \end{array}$$

We are given the goal[2]

$$?\vdash (\lambda x \bullet Sin(Cos(Exp\ x))) \in Cts$$

One's first thought might be to express the compositionality property as follows:

$$\vdash \forall f\ g \bullet\ f \in Cts \land g \in Cts \Rightarrow (\lambda x \bullet f(g\ x)) \in Cts$$

and then to attack the goal by backchaining[3]. I.e., one would first match the goal with

$$(\lambda x \bullet f(g\ x)) \in Cts$$

and use the compositionality property to reduce the problem to the subgoals

$$\begin{array}{l} ?\vdash Sin \in Cts \\ ?\vdash (\lambda x \bullet\ Cos(Exp\ x)) \in Cts \end{array}$$

then the former subgoal can be discharged as it is one of the ground facts and the latter can again be matched with $(\lambda x \bullet f(g\ x)) \in Cts$. Unfortunately, this involves general higher-order matching: the term $(\lambda x \bullet f(g\ x)) \in Cts$ is not a linear pattern[4] in the sense of Miller [Mil91] and so the efficient Miller-Nipkow algorithm [Nip93] does not apply. Although the general higher-order matching problem is known to be decidable [Sti09], it is also known to be of non-elementary complexity. We could write a custom function to deal with the particular cases of higher-order matching needed here, but we prefer an approach that makes best use of existing proof infrastructure (which includes first-order matching and higher-order matching for linear patterns). With this in mind, we express the compositionality property using the functional composition operator $\circ$:

[2]We write $\vdash \phi$ for a theorem that has been proved and $?\vdash \phi$ for a conjecture or goal. More generally, theorems and goals in HOL are sequents $\Gamma \vdash \phi$ and $\Gamma\, ?\vdash \phi$ where the assumptions Γ comprise a set of formulas, but we do not need sequents with assumptions here.
[3]We use the term *backchaining* to describe a class of proof techniques that use a stock of universally closed implications $\forall x_1, \ldots, x_m \bullet \phi_1 \land \ldots \land \phi_n \Rightarrow \psi$, where possibly $n = 0$ so that the "implication" is actually a ground fact. In a backchaining proof, given a goal $?\vdash \chi$, we attempt to match χ with the succedent ψ of each implication and when a substitution θ giving a match is found, we replace the goal with the subgoals obtained by existentially closing $(\phi_1 \land \ldots \land \phi_n)\theta$ with respect to any free variables that do not appear free in ψ and then carrying out logical simplifications such as pushing the existential quantifiers in through conjunctions and breaking conjunctions into multiple subgoals. We may also use application-specific heuristics. e.g., to choose existential witnesses.
[4]A *linear pattern* is a λ-term in β-normal form in which in every subterm of the form $f\, x_1\, \ldots\, x_n$ where f is a variable, the x_i are all η-equivalent to variables.

$$\vdash \forall f\ \ g\bullet\ f \in Cts \wedge g \in Cts \Rightarrow g\ \ o\ f \in Cts$$

and translate our original goal:

$$?\vdash\ (\lambda x\bullet Sin(Cos(Exp\ \ x))) \in\ Cts$$

into

$$?\vdash\ Sin\ \ o\ \ Cos\ \ o\ \ Exp\ \in\ Cts$$

The problem can then be solved just by backchaining using first-order matching: higher-order matching for linear patterns will be important in the sequel, but we do not need it yet.

2.3 Products

We say a concrete category $\mathcal{C}$ has *standard products* if the standard set-theoretic product construction yields a product in $\mathcal{C}$. That is to say, if X and Y are objects of $\mathcal{C}$, there is an object $X \times_{\mathcal{C}} Y$ of $\mathcal{C}$, such that *(i)* $U(X \times_{\mathcal{C}} Y) = U(X) \times U(Y)$, *(ii)* the projections $\pi_1 : U(X \times Y) \to U(X)$ and $\pi_2 : U(X \times Y) \to U(Y)$ are morphisms of $\mathcal{C}$ and *(iii)*, if $f : Z \to X$ and $g : Z \to Y$ are morphisms of $\mathcal{C}$, there is a unique morphism $\langle f, g\rangle : Z \to X \times_{\mathcal{C}} Y$ such that $f = \pi_1 \circ \langle f, g\rangle$ and $g = \pi_2 \circ \langle f, g\rangle$. (If one drops requirement *(i)*, this is the usual definition of a product.)

Many useful examples of concrete categories have standard products, e.g., **Top**, and any concrete category like the category of partially ordered groups that can be axiomatized by first-order Horn clauses [Hod93, Chapter 9]. For a simple example of a concrete category that has products but not standard products, choose some set U with at least two elements and consider the category $\mathcal{S}_U$ whose objects are subsets of U and whose morphisms are the inclusions. If X, Y and Z are objects of $\mathcal{S}_U$, there are morphisms from Z to X and from Z to Y iff $Z \subseteq X \cap Y$, hence X and Y have $X \cap Y$ as their product.

In our running example, the product structure on **Top** is given on objects by the product topology formalised in ProofPower as follows:

$$\$\times_T\ :\ 'a\ \mathbb{P}\ \mathbb{P} \to\ 'b\ \mathbb{P}\ \mathbb{P} \to (\ 'a\ \times\ 'b)\ \mathbb{P}\ \mathbb{P}$$

$$\begin{aligned}&\forall \sigma\ \tau\bullet\ \ (\sigma\ \times_T\ \tau) = \\ &\quad \{C \mid \forall\ x\ y\bullet\ (x,\ y) \in C \\ &\qquad \Rightarrow \exists A\ \ B\bullet\ A \in \sigma \wedge B \in \tau \wedge x \in A \\ &\qquad\quad \wedge\ y \in B \wedge (A\ \times\ B) \subseteq C\}\end{aligned}$$

The pairing operator $\langle f, g\rangle$ (which is the same in any concrete category with standard products) is provided already in the ProofPower-HOL library as if by the following:

$$Pair(\ 'a \to\ 'b)\ \times\ (\ 'a \to\ 'c) \to\ 'a \to\ 'b\ \times\ 'c$$

$$\forall\ f\ \ g\bullet\ Pair\ (f,\ g) = (\lambda\ x\bullet\ (f\ \ x,\ g\ \ x))$$

Similarly, a concrete category $\mathcal{C}$ has *standard sums* if the standard set-theoretic sum (i.e., disjoint union) yields a sum in $\mathcal{C}$. **Top** has standard sums, but categories of algebras typically do not: for example, in the category of abelian groups, finite sums are isomorphic to products: $X+Y \simeq X \times Y$. We will concentrate on products in the present paper.

2.4 Proving morphismhood in $(\mathbb{R}, \times; \langle\rangle, \mathsf{exp}, \mathsf{sin}, \mathsf{cos}, +, \times)_{\mathbf{Top}}$

Once we allow product topologies, we can construct an infinite number of distinct topological spaces from the ground space $\mathbb{R}$. We will therefore need to add the following general compositionality property to our stock of facts.

$$\begin{aligned}&\vdash \forall\ \rho\ \sigma\ \tau\ f\ g \bullet \rho \in \mathit{Topology} \land \sigma \in \mathit{Topology} \land \tau \in \mathit{Topology} \land\\ &\qquad\qquad f \in (\rho,\ \sigma)\ \mathit{Continuous} \land g \in (\sigma,\ \tau)\ \mathit{Continuous}\\ &\qquad \Rightarrow g\ o\ f \in (\rho,\ \tau)\ \mathit{Continuous}\end{aligned}$$

Note that when we match a subgoal such as

$$?\vdash \mathit{Sin}\ o\ \mathit{Cos} \in (O_R,\ O_R)\ \mathit{Continuous}$$

with the right-hand side of the implication in the above and then backchain, the subgoal we get is existentially quantified in the intermediate topology σ:

$$\begin{aligned}&?\vdash \exists\ \sigma\bullet\ \sigma \in \mathit{Topology} \land \mathit{Cos} \in (O_R,\ \sigma)\ \mathit{Continuous}\\ &\qquad \land \mathit{Sin} \in (\sigma,\ O_R)\ \mathit{Continuous}\end{aligned}$$

For the examples in this section, all topological spaces are obtained from the standard topology on $\mathbb{R}$ using the product topology construction, hence we can always select an appropriate existential witness in the above by inspecting the type of σ, which in this example is; $((\mathbb{R})\mathbb{P})\mathbb{P}$, the type of a topology on $\mathbb{R}$, so we will instantiate σ to the standard topology O_R.

Specific to products, we have the facts that the projection function, π_1 and π_2 (written *Fst* and *Snd* in ProofPower-HOL) are continuous and that the pairing of two continuous functions is continuous.

$$\begin{aligned}&\vdash \forall\ \sigma\ \tau\bullet\ \sigma \in \mathit{Topology} \land \tau \in \mathit{Topology}\\ &\qquad \Rightarrow \mathit{Fst} \in (\sigma \times_T \tau,\ \sigma)\ \mathit{Continuous}\\ &\vdash \forall\ \sigma\ \tau\bullet\ \sigma \in \mathit{Topology} \land \tau \in \mathit{Topology}\\ &\qquad \Rightarrow \mathit{Snd} \in (\sigma \times_T \tau,\ \tau)\ \mathit{Continuous}\\ &\vdash \forall\ \rho\ \sigma\ \tau\ f\ g\bullet\ \rho \in \mathit{Topology} \land \sigma \in \mathit{Topology} \land \tau \in \mathit{Topology} \land\\ &\qquad\qquad f \in (\rho,\ \sigma)\ \mathit{Continuous} \land g \in (\rho,\ \tau)\ \mathit{Continuous}\\ &\qquad \Rightarrow \mathit{Pair}\ (f,\ g) \in (\rho,\ \sigma \times_T \tau)\ \mathit{Continuous}\end{aligned}$$

To use the above, we will need to prove that various instances of the product topology are actually topologies, for which purpose we have the following fact:

$$\vdash \forall\ \sigma\ \tau\bullet\ \sigma \in \mathit{Topology} \land \tau \in \mathit{Topology} \Rightarrow \sigma \times_T \tau \in \mathit{Topology}:$$

Let us see how we might prove a goal such as the following using our augmented stock of facts,

$$?\vdash (\lambda x\bullet(\mathit{Sin}(\mathit{Exp}\ x),\ \mathit{Cos}\ (\mathit{Exp}\ x))) \in (O_R,\ O_R \times_T O_R)\ \mathit{Continuous}$$

To apply our facts, we first translate the λ-abstraction in our goal using the composition and pairing combinators to give:

$$?\vdash \mathit{Pair}\ (\mathit{Sin}\ o\ \mathit{Exp}\ ,\ \mathit{Cos}\ o\ \mathit{Exp}) \in (O_R,\ O_R \times_T O_R)\ \mathit{Continuous}$$

Back-chaining with our facts now solves the problem. Of course to automate the process we have just described, we need to automate the translation of the λ-abstraction into the combinator form. We will describe a general scheme for this in the next section.

Our general translation scheme will also allow us to prove continuity of expressions involving binary operators such as addition. For this, we need to address the minor detail that ProofPower-HOL follows many type-theoretic formalisation of mathematics in defining these as curried functions: they have type $\mathbb{R} \to \mathbb{R} \to \mathbb{R}$. So, for example, we translate the λ-abstraction:

$$?\vdash (\lambda(x,\ y)\bullet\ Exp(x\ +\ y)) \in (O_R \times_T O_R,\ O_R)\ Continuous$$

into the combinator form:

$$?\vdash Exp\ o\ Uncurry\ \$+\ o\ Pair\ (Fst,\ Snd) \\ \in (O_R \times_T O_R,\ O_R)\ Continuous$$

where the function *Uncurry* converts a two-place function into the corresponding function on pairs. Given the additional fact:

$$\vdash Uncurry\ \$+ \in (O_R \times_T O_R,\ O_R)\ Continuous$$

backchaining with our facts will yet again solve the goal.

With just a little more work, we can deal with arbitrary exponential/trigonometric polynomials. We have just to extend our approach to cover constant functions, negation, multiplication and exponentiation with natural number exponents. For these consider the following example:

$$?\vdash (\lambda x\bullet\ 2.\ *\ \sim (x\ \hat{}\ 4)) \in (O_R,\ O_R)\ Continuous$$

Here, $\sim\ : \mathbb{R} \to \mathbb{R}$ is negation and $\hat{}\ : \mathbb{R} \to \mathbb{N} \to \mathbb{R}$ is the exponentiation operator, which we think of as a family of continuous functions parametrized by a natural number. We translate the goal into:

$$?\vdash Uncurry\ \$*\ o\ Pair\ (CombK\ 2.,\ \sim\ o\ (\lambda\ x\bullet\ x\ \hat{}\ 4)) \\ \in (O_R,\ O_R)\ Continuous$$

where *CombK* is the ProofPower-HOL name for the K combinator. This may be solved by backchaining using the new facts:

$$\begin{array}{l} \forall\ \sigma\ \tau\ c\bullet\ \sigma \in Topology \land \tau \in Topology \land c \in Space_T\ \tau \\ \quad \Rightarrow CombK\ c \in (\sigma,\ \tau)\ Continuous \\ \vdash Uncurry\ \$* \in (O_R \times_T O_R,\ O_R)\ Continuous \\ \vdash \sim\ \in (O_R,\ O_R)\ Continuous \\ \vdash \forall n\bullet\ (\lambda\ x\bullet\ x\ \hat{}\ n) \in (O_R,\ O_R)\ Continuous \\ \vdash \forall x\bullet\ x \in Space_T\ O_R \end{array}$$

As a final example for this section, let us consider the famous complex function $f(z) = e^{2\pi i z}$. In the ProofPower-HOL formalisation, the goal stating that this function is continuous is as follows:

$$(\lambda z\bullet\ Exp(\mathbb{RC}\ 2.\ *\ \mathbb{RC}\ \pi\ *\ I_C\ *\ z)) \in (O_C,\ O_C)\ Continuous$$

where:

—*Exp* is an (overloaded) alias for the complex exponential function defined (using its real name Exp_C) as follows:

$$\begin{array}{|l} Exp_C : \mathbb{C} \to \mathbb{C} \\ \hline \forall z \bullet\ Exp_C\ z = \mathbb{RC}\ (Exp(Re\ z)) * (Cos\ (Im\ z),\ Sin\ (Im\ z)) \end{array}$$

—$\mathbb{C}$ is an abbreviation for the type $\mathbb{R} \times \mathbb{R}$,
—$\mathbb{RC}$ is the inclusion of $\mathbb{R}$ as a subfield of $\mathbb{C}$ defined by $\forall x \bullet\ \mathbb{RC}\ x = (x,\ 0.)$,
—$I_C = (0.,\ 1.)$ is the complex number i,
—*Re* and *Im* are aliases for the $\mathbb{C} \to \mathbb{R}$ instances of *Fst* and *Snd* respectively,
—$*$ is an (overloaded) alias for multiplication of complex numbers, and
—the exponential and trigonometric functions on the right-hand side of the equation have type $\mathbb{R} \to \mathbb{R}$.

The goal translates into combinator form as follows:

```
?⊢ Pair (
     Uncurry $* o Pair (
          Exp o Uncurry $* o Pair (
               CombK 2.,
               Uncurry $* o Pair (CombK π, ~ o Im)),
          Cos o Uncurry $* o Pair (
               CombK 2.,
               Uncurry $* o Pair (CombK π, Re))),
     Uncurry $* o Pair (
          Exp o Uncurry $* o Pair (
               CombK 2.,
               Uncurry $* o Pair (CombK π, ~ o Im)),
          Sin o Uncurry $* o Pair (
               CombK 2.,
               Uncurry $* o Pair (CombK π, Re))))
                                        ∈ (O_C, O_C) Continuous
```

Note that the complexity of this formula arises because we are here treating $\mathbb{C}$ as the $\mathbb{R}$-algebra $\mathbb{R}[i]$, in order to view its topology as the topology of the plane $\mathbb{R} \times \mathbb{R}$. After expansion of the definition of the topology on the complex plane, namely $O_C = O_R \times O_R$, backchaining with our existing stock of facts yet again solves this goal.

2.5 Representing other categories

A concrete category such as **Grp** defined by first-order axioms over a finite signature is naturally formalised in HOL as a subclass of a polymorphic labelled product type with a component defining the underlying set of an object and a component for each function or predicate symbol in the signature. For example, in ProofPower-HOL, groups are formalised as a subclass of a labelled product with four components defined using the following syntax:

HOL Labelled Product

```
┌GROUP─────────────────────────────────
│   Car_G        : 'a ℙ;
│   Times_G      : 'a → 'a → 'a;
│   Unit_G       : 'a;
│   Inverse_G    : 'a → 'a
└──────────────────────────────────────
```

The above syntax introduces a polymorphic labelled product type $('a)GROUP$ with components of the indicated names and types. The component Car_G denotes the underlying set of the group (a.k.a., its carrier set) and the remaining components represent the multiplication, the identity element and the inverse function. Groups are then defined by a polymorphic subclass *Group* of the type $('a)GROUP$ comprising the labelled 4-tuples that satisfy the group axioms. As with continuous functions on topological spaces, group homomomorphisms are represented by functions $('a)GROUP \to ('b)GROUP$ whose restrictions to the carrier set of the domain respect the group structure but whose behaviour outside the carrier set is irrelevant. We define $Homomorphis(G,\ H)$ to be the set of homorphisms from G to H.

2.6 Proving morphismhood in $(\mathbb{R}_+, \mathbb{C}_+, \mathbb{C}_\times; \exp, c \times \_)_{\mathbf{Grp}}$

The assertion that the function $f(x) = e^{2\pi ix}$ defines a homomorphism from the additive group of real numbers to the multiplicative group of positive complex numbers is formalised as the following goal:

$$\begin{aligned}?\vdash\ & (\lambda x\bullet\ Exp(\mathbb{RC}\ 2.\ *\ \mathbb{RC}\ \pi\ *\ I_C\ *\ \mathbb{RC}\ x))\\ & \in Homomorphism\ (\mathbb{R}_+,\ \mathbb{C}_*)\end{aligned}$$

If we translate the goal into:

$$\begin{aligned}?\vdash\ & Exp\ o\ \$*\ (\mathbb{RC}\ 2.)\ o\ \$*\ (\mathbb{RC}\ \pi)\ o\ \$*\ I_C\ o\ \mathbb{RC}\\ & \in Homomorphism\ (\mathbb{R}_+,\ \mathbb{C}_*)\end{aligned}$$

the goal will follow by backchaining using the following facts:

$$\begin{aligned}&\vdash Exp \in Homomorphism\ (\mathbb{C}_+,\ \mathbb{C}_*);\\ &\vdash \mathbb{RC} \in Homomorphism\ (\mathbb{R}_+,\ \mathbb{C}_+);\\ &\vdash \forall\ c : \mathbb{C}\bullet\ \$*\ c \in Homomorphism\ (\mathbb{C}_+,\ \mathbb{C}_+):\\ &\vdash \forall\ G\ H\ K\ f\ g\bullet\\ &\qquad\qquad G \in Group \wedge H \in Group \wedge K \in Group\\ &\qquad \wedge \quad f \in Homomorphism\ (G,\ H) \wedge g \in Homomorphism\ (H,\ K)\\ &\qquad \Rightarrow \quad g\ o\ f \in Homomorphism\ (G,\ K)\\ &\vdash \mathbb{R}_+ \in Group\\ &\vdash \mathbb{C}_+ \in Group\\ &\vdash \mathbb{C}_* \in Group\end{aligned}$$

3. THE PROOF PROCEDURE

The examples of the previous section suggest a proof procedure for proving morphismhood in a particular concrete category. The goals tackled by the proof procedure have the form

$$? \vdash f \in \mathsf{Mor}_{\mathcal{C}}(X, Y)$$

where:

—$\mathcal{C}$ is the concrete category of interest which is assumed to have standard products[5],

—f is a λ-abstraction representing a function that we wish to show is a morphism of $\mathcal{C}$,

—X and Y are expressions formed from some given objects of $\mathcal{C}$ using the product in $\mathcal{C}$.

The proof procedure translates f into a combinator form and then attempts to solve the resulting goal by backchaining with theorems asserting that the combinators map morphisms to morphisms and that the given morphisms and objects are indeed morphisms and objects of $\mathcal{C}$. These theorems may impose additional conditions, e.g., that some expression belongs to the underlying set of an object.

We describe these two phases of the proof procedure in detail in sections 3.1 and 3.2 below.

3.1 Translation to combinator form

We will work in typed λ-calculus with paired abstractions and projection operators. In this language we may write the addition operation on the real numbers either using a paired abstraction: $\lambda\,(x : \mathbb{R}, y : \mathbb{R}) \bullet x + y$ or using the projections: $\lambda\, p : \mathbb{R} \times \mathbb{R} \bullet \pi_1(p) + \pi_2(p)$. This language is supported as derived syntax in all the HOL implementation using Milner-Hindley type inference to allow the user to omit inessential type constraints. For present purposes it is convenient to take this language as primitive. In the abstract syntax of the language all type constraints are explicit and and all type operators are written postfix ($(\alpha, \beta)\!\rightarrow$ rather than $\alpha \rightarrow \beta$). The abstract syntax is thus given by the following grammar:

$$\begin{array}{lcl}
\mathsf{Type} & = & \mathsf{TyVar} \mid (\mathsf{Type}, ..., \mathsf{Type})\mathsf{TyOp} \\
\mathsf{Pat} & = & \mathsf{Var} : \mathsf{Type} \mid (\mathsf{Pat}, \mathsf{Pat}) \\
\mathsf{Term} & = & \mathsf{Var} : \mathsf{Type} \mid \mathsf{Con} : \mathsf{Type} \mid \mathsf{Term}\,\mathsf{Term} \mid \lambda\mathsf{Pat}\bullet\,\mathsf{Term}
\end{array}$$

Here TyVar and Var denote two infinite set of variables names. The set $\mathsf{vars}(p) \subseteq \mathsf{Var} \times \mathsf{Type}$ of typed variables that occur in a pattern p is defined in the obvious way. Patterns and terms are subject to the usual rules that assign a unique type to each well-typed pattern or term. See [FL96] for details. In particular, a pattern (p_1, p_2) is only well-typed if $\mathsf{vars}(p_1) \cap \mathsf{vars}(p_2) = \varnothing$, in which case its type is $\tau_1 \times \tau_2$ where τ_i is the type of p_i, $i = 1, 2$.

We assume given a *signature* Σ defining the types and constants that we may use: it gives the arity of each type operator and the polymorphic type of each constant. Σ will include at least the (infix) binary type operators $\rightarrow$ and $\times$ and the following

[5]The procedure is easily adapted to work for categories that do not have standard products just by not providing the theorems and other parameters that relate to products.

constants (where α, β and γ are type variables):

$$\begin{array}{rcl}
\mathsf{I} & : & \alpha \to \alpha \\
\_ \circ \_ & : & (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma \\
\pi_i & : & \alpha \times \beta \to \alpha \\
\pi_2 & : & \alpha \times \beta \to \beta \\
\langle \_, \_ \rangle & : & (\alpha \to \beta) \to (\alpha \to \gamma) \to \alpha \to \beta \times \gamma \\
(\_, \_) & : & \alpha \to \beta \to \alpha \times \beta \\
\mathsf{K} & : & \alpha \to \beta \to \alpha \\
\mathsf{Uncurry} & : & (\alpha \to \beta \to \gamma) \to \alpha \times \beta \to \gamma
\end{array}$$

The translation into combinator form is parametrized by three sets of terms called unary, binary and parametrized that denote morphisms of $\mathcal{C}$ represented in three different ways: unary represents morphisms represented directly as functions, e.g., exp as a morphism $\mathbb{R} \to \mathbb{R}$ in **Top**; binary represents morphisms on product objects represented as curried functions e.g., multiplication represented with type $\mathbb{R} \to \mathbb{R} \to \mathbb{R}$ rather than $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$; parametrized represents a family of objects parametrized by a parameter passed as the second argument to a curried function, e.g., exponentiation of real numbers by natural number coefficients represented with type $\mathbb{R} \to \mathbb{N} \to \mathbb{R}$. Note that a family of objects parametrized by a parameter passed as the first argument to a curried function, e.g., $c \times \_ : \mathbb{R} \to \mathbb{R}$ can be represented by including all their instances in the set unary.

The translation is defined by repeated application of a set of rewrite rules defined as follows, where p is a pattern, where x and y are typed variables ($\mathsf{Var} : \mathsf{Type}$), c is a typed constant ($\mathsf{Con} : \mathsf{Type}$), where other metavariables are arbitrary terms of the appropriate types meeting any side-conditions stated and where π^p_x is I, if $p = x$, and is the composite of instances of π_1 and π_2 that extracts x from p, if $p \neq x \in \mathsf{vars}(p)$, e.g., $\pi^{((x_1,y_1),(x_2,y_2))}_{y_1} = \pi_2 \circ \pi_1$.

$$\begin{array}{rcll}
(\lambda p \bullet x) & \rightsquigarrow & \pi^p_x & x \in \mathsf{vars}(p) \\
(\lambda p \bullet y) & \rightsquigarrow & \mathsf{K}\, y & y \notin \mathsf{vars}(p) \\
(\lambda p \bullet c) & \rightsquigarrow & \mathsf{K}\, c & \\
(\lambda p \bullet (t_1, t_2)) & \rightsquigarrow & \langle (\lambda p \bullet t_1), (\lambda p \bullet t_2) \rangle & \\
(\lambda p \bullet f\, t) & \rightsquigarrow & f \circ (\lambda p \bullet t) & f \in \mathsf{unary} \\
(\lambda p \bullet g\, t_1\, t_2) & \rightsquigarrow & \mathsf{Uncurry}\, g \circ \langle (\lambda p \bullet t_1), (\lambda p \bullet t_2) \rangle & g \in \mathsf{binary} \\
(\lambda p \bullet h\, t\, j) & \rightsquigarrow & (\lambda x \bullet h\, x\, j) \circ (\lambda p \bullet t) & h \in \mathsf{parametrized}
\end{array}$$

We will assume that the three sets of terms unary, binary and parametrized are pairwise disjoint, that no instance of $(\_, \_)$ or of $(\_, \_)(x)$ belongs to any of the three sets and that no term in unary has the form $s\, t$ where $s \in \mathsf{binary} \cup \mathsf{parametrized}$. These assumptions ensure that the rewrite rules are mutually exclusive. We will also assume that no term in any of the three sets contains a subterm that is a λ-abstraction or an instance of K, since that makes it easier to describe the properties of the rewrite system and is not a significant shortcoming in practice.

Let E be the smallest set of terms that contains all variables and constants that are not subterms of any of the terms in the sets unary, binary or parametrized and is such that whenever $t_1, t_2 \in E$, then so also are (t_1, t_2) and any of the applications $f\, t_1$, $g\, t_1\, t_2$ and $h\, t_1\, j$ that are well-typed, where $f \in \mathsf{unary}$, $g \in \mathsf{binary}$, $h \in \mathsf{parametrized}$ and j is any constant. If t is in E and p is any pattern, then it is clear that $\lambda p\bullet\, t$ can be rewritten using the above rules to a term t' that can be built from ground terms of the forms I, π_1, π_2, $\mathsf{K}\,y$, $\mathsf{K}\,c$, f, $\mathsf{Uncurry}\,g$ and $\lambda x\bullet\, h\,x\,j$ using composition $\_ \circ \_$ and pairing $(\_, \_)$[6].

We now have to consider how to implement the rewrite system in practice. What we will typically want to represent in the sets unary and binary includes *(i)* specific morphisms, e.g., the monomorphic constant $Exp\ :\ \mathbb{C}\ \to\ \mathbb{C}$ as a group homomorphism from $\mathbb{C}_+$ to $\mathbb{C}_*$, *(ii)* families of morphisms defined by type instantiation, e.g., the set of all type instances of the polymorphic constant $Fst\ :\ 'a\ \times\ 'b\ \to\ 'a$ as morphisms in any concrete category with standard products, *(iii)* families of morphisms defined by term instantiation, e.g., all instances of $\$*\ z\ :\ \mathbb{C}\ \to\ \mathbb{C}$ as group homomorphisms from $\mathbb{C}_+$ to $\mathbb{C}_+$. We may also have variables in the sets that we do not want to instantiate, e.g., coming from background assumptions. What we will want to include in parametrized will be families of morphisms parametrized by their second argument, such as $\widehat{}\ :\ \mathbb{R}\ \to\ \mathbb{N}\ \to\ \mathbb{R}$. A suitable representation of each of these sets is therefore as a list of terms, which we will call the *unary, binary* and *parametrized operators* together with a list of term variables that are subject to instantiation. (The term variables not in the list of instantiable variables will effectively be treated as constants by the proof procedure).

The rules of the rewrite system are all instances of higher-order equations that we can prove in advance. We call such preproved theorems that capture the validity of a step in a proof procedure *template theorems.* Typically, template theorems are instantiated as necessary and used as rewrite rules. For example, the rule for binary is justified by the following template theorem.

$$\vdash \forall f\ \ s\ \ t\bullet\ (\lambda x\bullet f\ \ (s\ \ x)\ \ (t\ \ x)) = Uncurry\ \ f\ \ o\ \ Pair(s,\ t)$$

Now the left-hand side of this template theorem is not a linear pattern, since f has the application $s\,x$ for its first argument. Fortunately, when we instantiate f to a constant (or to a constant applied to one or more linear patterns), the left-hand side becomes a linear pattern. For example, when we instantiate f to $+$, we get:

$$\vdash \forall s\ \ t\bullet\ (\lambda x\bullet(s\ \ x)\ +\ (t\ \ x)) = Uncurry\ \ \$+\ \ o\ \ Pair(s,\ t)$$

Thus the instantiation of the template theorem is a two-stage process: it is instantiated once for each binary operator resulting in a form that is suitable for our higher-order matcher, which we then use repeatedly to find the instantiations needed to process any given input goal. Similarly, the template theorem justifying the rule for unary is:

$$\vdash \forall f\ \ t\bullet\ (\lambda x\bullet f\ \ (t\ \ x)) = f\ \ o\ \ t$$

The instantiation of this to represent the family of homomorphisms from $\mathbb{C}_+$ to $\mathbb{C}_+$ given by left-multiplication is as follows:

$$\vdash \forall t\bullet\ (\lambda x\bullet\ \$*\ \ z\ \ (t\ \ x)) = \$*\ \ z\ \ o\ \ t$$

The rewrite rules for unary, binary and parametrized can therefore be implemented by instantiating the template theorem to the various operators and then arranging

[6]In particular, the ground terms $\lambda x\bullet\, h\,x\,j$ are the only λ-abstractions in t'. A case could be made for replacing $\lambda x\bullet\, h\,x\,j$ in the rewrite rules by $\mathsf{switch}\,h\,p$ where $\mathsf{switch}\,f\,x\,y = f\,y\,x$, so there would be no λ-abstractions at all. The description here matches what has actually been implemented.

for instantiation to be enabled during rewriting for the instantiable variables only (in ProofPower this may be achieved just by forming the universal closure with respect to the instantiable variables).

Thus we can arrange for all the instances of the rewrite rules that we need to be expressed as linear patterns. In order to use the Miller-Nipkow higher-order matching algorithm [Nip93], we also need to transform the abstraction over a pattern p into abstraction over a plain variable[7]. This is straightforward, e.g., $(\lambda(x,\ y) \bullet x\ +\ y)$ can be preprocessed into $\lambda xy \bullet\ \mathit{Fst}\ xy\ +\ \mathit{Snd}\ xy.$ using a conversion provided as part of the ProofPower proof infrastructure.

3.2 Proving the combinator form is a morphism

We now consider the problem of proving that the combinator form produced by the first phase of our proof procedure is actually a morphism in the category $\mathcal{C}$. We can assume that the goal now has the form $?\vdash (\lambda x \bullet\, t) \in \mathsf{Mor}_{\mathcal{C}}(X, Y)$, where t is formed from given morphisms using composition $\_ \circ \_$ and pairing $\langle \_, \_ \rangle$ and where X and Y are formed from given objects of $\mathcal{C}$ using binary products. We assume we have proved theorems asserting that that given morphisms are indeed morphisms, that that the given objects are indeed objects together with the following theorems allowing us to construct new objects and morphisms from old:

$$\vdash \forall X\, Y \bullet X \in \mathsf{Obj}_{\mathcal{C}} \wedge Y \in \mathsf{Obj}_{\mathcal{C}} \Rightarrow X \times_{\mathcal{C}} Y \in \mathsf{Obj}_{\mathcal{C}}$$

$$\begin{aligned} &\vdash \forall X\, Y\, Z\, f\, g \bullet \\ &\qquad X \in \mathsf{Obj}_{\mathcal{C}} \wedge Y \in \mathsf{Obj}_{\mathcal{C}} \wedge Z \in \mathsf{Obj}_{\mathcal{C}} \wedge f \in \mathsf{Mor}_{\mathcal{C}}(X, Y) \wedge g \in \mathsf{Mor}_{\mathcal{C}}(X, Z) \\ &\ \Rightarrow\ \langle f, g \rangle \in \mathsf{Mor}_{\mathcal{C}}(X, Y \times_{\mathcal{C}} Z) \end{aligned}$$

$$\begin{aligned} &\vdash \forall X\, Y\, Z\, f\, g \bullet \\ &\qquad X \in \mathsf{Obj}_{\mathcal{C}} \wedge Y \in \mathsf{Obj}_{\mathcal{C}} \wedge Z \in \mathsf{Obj}_{\mathcal{C}} \wedge f \in \mathsf{Mor}_{\mathcal{C}}(X, Y) \wedge g \in \mathsf{Mor}_{\mathcal{C}}(Y, Z) \\ &\ \Rightarrow\ g \circ f \in \mathsf{Mor}_{\mathcal{C}}(X, Z) \end{aligned}$$

Backchaining with first-order matching, e.g., as implemented in the ProofPower tactic *bc_tac* is exactly what we need to use these theorems. However, backchaining with the theorem about $\_ \circ \_$ will produce a subgoal of the form:

$$?\vdash \exists Y \bullet X \in \mathsf{Obj}_{\mathcal{C}} \wedge Y \in \mathsf{Obj}_{\mathcal{C}} \wedge Z \in \mathsf{Obj}_{\mathcal{C}} \wedge f \in \mathsf{Mor}_{\mathcal{C}}(X, Y) \wedge g \in \mathsf{Mor}_{\mathcal{C}}(Y, Z)$$

We need some way of choosing the right intermediate object Y. A simple but effective heuristic for this is to use the type of the term Y to decide what object to use. Typically, we will only have a finite number of relevant objects of $\mathcal{C}$ of a given HOL type, so we can use the type to select a list of possibilities.

If we want to package the proof procedure as a decision procedure, then we can do a search with backtracking for the right choice of objects. However, we may also want to package the proof procedure as a heuristic for interactive use that may make useful progress on a goal without solving it completely. For example, if A and B are topological spaces, the function $\mathsf{K}\, x$ does not define a continuous

[7]This would be unnecessary given an implementation of the matching algorithm of Fettig and Löchner which handles paired abstractions and projections [FL96]. However, the special case that is of interest here does not require the full generality.

function from A to B unless $x \in B$, and this might require a proof that is outside the scope of our morphismhood proof procedure. A backtracking approach would be inappropriate when the proof procedure is used in this way.

In practice, at least when working with abstract topological spaces, the proof procedure has proved very useful just using the naive heuristic of picking the first known topology that has the right type. As an example where this heuristic fails, consider the following problem about group homomorphisms:

$$?\vdash (\lambda x\bullet\ \mathit{Exp}(x)^{-}) \in \mathit{Homomorphism}(\mathbb{C}_{+},\ \mathbb{C}_{*})$$

Here the postfix operator written z^{-} denotes conjugation of complex numbers (which is a homomorphism on both $\mathbb{C}_{+}$ and $\mathbb{C}_{*}$). It turns out that our implementation of the naive heuristic will lead to the false subgoals:

$$?\vdash \$^{-} \in \mathit{Homomorphism}\ (\mathbb{C}_{+},\ \mathbb{C}_{*})$$
$$?\vdash \mathit{Exp} \in \mathit{Homomorphism}\ (\mathbb{C}_{+},\ \mathbb{C}_{+})$$

The one-object-per-type approach has picked the wrong intermediate group. A backtracking implementation of the proof procedure as a decision procedure will try again and find the correct intermediate subgoals:

$$?\vdash \$^{-} \in \mathit{Homomorphism}\ (\mathbb{C}_{*},\ \mathbb{C}_{*})$$
$$?\vdash \mathit{Exp} \in \mathit{Homomorphism}\ (\mathbb{C}_{+},\ \mathbb{C}_{*})$$

and so solve the goal. Since proofs in which there are many different group structures on the same underlying set are rare, a naive backtracking approach should be relatively efficient in most cases.

4. IMPLEMENTATION AND APPLICATIONS

A prototype of the generic proof procedure described in section 3 has been implemented and applied in the ProofPower-HOL mathematical case studies [Art14]. A reader who downloads the case study source will find the implementation of the generic procedure (`basic_morphism_tac`) in the file `wrk083.doc` and instantiations (`basic_continuity_tac`, $\mathbb{R}$`_continuity_tac`) in the file `wrk067.doc`.

A design goal was to make good use of the existing matching, rewriting and backchaining tools and we believe this was achieved: the implementation of the basic proof procedure is less than 100 lines of Standard ML code. This is augmented by about 30 lines of interface code which automates the process of deriving the sets unary, binary and parametrized by analysing a set of theorems asserting the necessary categorical properties of some collection of objects, morphisms and operators thereon.

The generic proof procedure is implemented as a tactic that can make partial progress and then hand the problem back to the user. However, the interface code is relatively simplistic and, in particular, does not currently handle arbitrary side-conditions in these theorems. This means that, in practice, when the procedure fails it tends to be because a necessary assumption is missing and the solution is to provide that assumption and then try again.

The proof procedure has been found to be a very convenient tool in the formalisation of some of the basics of abstract topology, metric spaces, homotopy theory and group theory and in putting this theory to work on some of the standard examples of functions on the complex plane with interesting topological and algebraic properties.

Here, for example, are the definitions of the notion of homotopy and a homotopy class used in the case studies. As a technical convenience, a homotopy between functions $f, g : S \to T$ relative to a subspace X of S is taken to be a continuous function from $S \times \mathbb{R} \to T$ that agrees with f (resp. g) when restricted to $S \times \{0\}$ (resp. $T \times \{1\}$). (This is equivalent to the standard definition of a homotopy as a continuous function on $S \times [0, 1]$, because such a function can always be extended to $S \times \mathbb{R}$.)

$$\begin{array}{|l}
\$\mathit{Homotopy} : {'a}\ \mathbb{P}\ \mathbb{P} \times {'a}\ \mathbb{P} \times {'b}\ \mathbb{P}\ \mathbb{P} \to ({'a} \times \mathbb{R} \to {'b})\ \mathbb{P} \\
\hline
\forall \sigma\ X\ \tau \bullet (\sigma,\ X,\ \tau)\ \mathit{Homotopy} = \\
\quad \{\ f \\
\quad |\quad f \in ((\sigma \times_T O_R),\ \tau)\ \mathit{Continuous} \\
\quad \wedge\quad \forall x\ s\ t \bullet x \in X \Rightarrow f(x,\ s) = f(x,\ t)\}
\end{array}$$

$$\begin{array}{|l}
\$\mathit{HomotopyClass} : {'a}\ \mathbb{P}\ \mathbb{P} \times {'a}\ \mathbb{P} \times {'b}\ \mathbb{P}\ \mathbb{P} \to ({'a} \to {'b}) \to ({'a} \to {'b})\ \mathbb{P} \\
\hline
\forall \sigma\ X\ \tau\ f \bullet ((\sigma,\ X,\ \tau)\ \mathit{HomotopyClass})\ f = \\
\quad \{g \\
\quad |\ \exists H \bullet\ H \in (\sigma,\ X,\ \tau)\ \mathit{Homotopy} \\
\quad \wedge\ (\forall x \bullet\ H(x,\ \mathbb{NR}\ \mathit{0}) = f\ x) \wedge (\forall x \bullet\ H(x,\ \mathbb{NR}\ \mathit{1}) = g\ x)\}
\end{array}$$

The case studies include proofs of basic facts about these notions, e.g. the proofs of the following theorem, whcih stat that the homotopy classes are the equivalence classes of an equivalence relation:

$$\begin{array}{l}
\vdash \forall\ \sigma\ X\ \tau\ f\ \bullet \\
\qquad\qquad \sigma \in \mathit{Topology} \wedge \tau \in \mathit{Topology} \\
\qquad \wedge\quad f \in (\sigma,\ \tau)\ \mathit{Continuous} \\
\qquad \Rightarrow\quad f \in ((\sigma,\ X,\ \tau)\ \mathit{HomotopyClass})\ f
\end{array}$$

$$\begin{array}{l}
\vdash \forall\ \sigma\ X\ \tau\ f\ g\ \bullet \\
\qquad\qquad \sigma \in \mathit{Topology} \wedge \tau \in \mathit{Topology} \\
\qquad \wedge\quad g \in ((\sigma,\ X,\ \tau)\ \mathit{HomotopyClass})\ f \\
\qquad \Rightarrow\quad f \in ((\sigma,\ X,\ \tau)\ \mathit{HomotopyClass})\ g
\end{array}$$

$$\begin{array}{l}
\vdash \forall\ \sigma\ X\ \tau\ f\ g\ h\ \bullet \\
\qquad\qquad \sigma \in \mathit{Topology} \wedge \tau \in \mathit{Topology} \\
\qquad \wedge\quad g \in ((\sigma,\ X,\ \tau)\ \mathit{HomotopyClass})\ f \\
\qquad \wedge\quad h \in ((\sigma,\ X,\ \tau)\ \mathit{HomotopyClass})\ g \\
\qquad \Rightarrow\quad h \in ((\sigma,\ X,\ \tau)\ \mathit{HomotopyClass})\ f
\end{array}$$

Let us look at the goal resulting from expanding the definition of *HomotopyClass* in the proof that the homotopy relation is symmetric (the second of the above three theorems). The goal is printed by ProofPower as a sequent with 6 assumptions in the context and a single conclusion:

$$\begin{array}{l}
(*\ \ 6\ *)\ulcorner \sigma \in \mathit{Topology}\urcorner \\
(*\ \ 5\ *)\ulcorner \tau \in \mathit{Topology}\urcorner \\
(*\ \ 4\ *)\ulcorner H \in (\sigma \times_T O_R,\ \tau)\ \mathit{Continuous}\urcorner \\
(*\ \ 3\ *)\ulcorner \forall\ x\ s\ t\bullet\ x \in X \Rightarrow H\ (x,\ s) = H\ (x,\ t)\urcorner \\
(*\ \ 2\ *)\ulcorner \forall\ x\bullet\ H\ (x,\ \mathit{0.}) = f\ x\urcorner \\
(*\ \ 1\ *)\ulcorner \forall\ x\bullet\ H\ (x,\ \mathit{1.}) = g\ x\urcorner
\end{array}$$

$$\begin{array}{l}
(*\ ?\vdash\ *)\ulcorner \exists\ H \\
\quad \bullet\ (H \in (\sigma \times_T O_R,\ \tau)\ \mathit{Continuous} \\
\qquad \land\ (\forall\ x\ s\ t\bullet\ x \in X \Rightarrow H\ (x,\ s) = H\ (x,\ t))) \\
\qquad \land\ (\forall\ x\bullet\ H\ (x,\ \mathit{0.}) = g\ x) \\
\qquad \land\ (\forall\ x\bullet\ H\ (x,\ \mathit{1.}) = f\ x)\urcorner
\end{array}$$

The 6 assumptions give us that H is a homotopy from f to g relative to X with respect to the topology σ on the domain of f and g and the topology τ on their range. We are asked to provide a homotopy from g to f. We supply the witness $\lambda\ xt \bullet\ H(\mathit{Fst}\ xt,\ \mathbb{NR}\ \mathit{1}\ -\ \ \mathit{Snd}\ xt)$ and a rewriting tactic then does standard algebraic simplifications using the assumptions to give us the same list of assumptions and the following conclusion left to prove:

$$\begin{array}{l}
(*\ ?\vdash\ *)\ulcorner(\lambda\ xt\bullet\ H\ (\mathit{Fst}\ xt,\ \mathit{1.} + \sim (\mathit{Snd}\ xt))) \\
\qquad\qquad\qquad \in (\sigma \times_T O_R,\ \tau)\ \mathit{Continuous}\urcorner
\end{array}$$

An application of `ℝ_continuity_tac` concludes the proof.

Deeper topological properties in the case studies include the unique lifting property and the homotopy lifting property for covering projections. The proof are based on proofs given in [May99] and [Spa95].

The exponential mapping from the real line, $\mathbb{R}$, to the unit circle S^1 in the complex plane is formally defined in the case studies as follows:

$$\begin{array}{|l}
\boldsymbol{ExpS1} : \mathbb{R} \to \mathbb{C} \\
\hline
\forall t\bullet\ \mathit{ExpS1}\ t = (\mathit{Cos}\ t,\ \mathit{Sin}\ t)
\end{array}$$

The proof that this is a group homomorphism from $\mathbb{R}_+$ to $\mathbb{C}_\times$ follows by easy algebra from the laws for sines and cosines of sums and does not need the machinery of this paper. However the proof that it is a covering projection makes extensive use of the proof procedure for proving continuity and would have been tedious to find without that. In total (at the time of writing), the proof procedure is used 5 times in the proofs about abstract topology, 22 times in proofs about homotopy and 13 times in the proofs about topological properties of the complex numbers.

5. CONCLUDING REMARKS

We believe that success in mechanizing a large corpus of mathematics is crucially dependent on a powerful array of tools to automate common proof tasks in an interactive theorem proving system. Proving morphismhood in typical formalisations of specific concrete categories like the categories of topological spaces, groups, etc.

is a very common task. We have presented a proof procedure to assist with this problem that has been implemented in ProofPower and has proved very useful. The basic ingredients of the proof procedure are the well-known Miller-Nipkow higher-order matching for linear patterns and backchaining with Horn clauses. The procedure will therefore admit a straightforward implementation in many programmable theorem-proving systems, for example, the various HOL systems or other members of the LCF family.

It is noteworthy that the categories of interest in this paper have a different flavour from the ones more commonly considered in computer science. Our proof procedure uses a rewrite system to convert a λ-term into a combinator form. The rewrite system is similar in spirit to bracket abstraction [Tur79]. However bracket abstraction makes essential use of the S combinator, which in a typed setting has polymorphic type $(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$. In a concrete category that is not cartesian closed, the first parameter of S will not in general be the type of any morphism in the category. Of the usual categories in which pure mathematics is conducted, very few (apart from **Set**) are cartesian closed. Non-trivial algebraic categories like **Grp** that have an initial object that is also terminal cannot be cartesian closed. **Top** comes closer to being cartesian closed, but while there are many ways of deriving a topology on $X \to Y$ from topologies on X and Y, pathological cases prevent any such topology from making **Top** into a cartesian closed category with respect to the usual product topology [EH02].

I believe an achievable goal in the not too distant future is a synthesis of ideas from automated and interactive theorem-proving, computation logic and computational algebra to provide the computational power and ease of use computer algebra systems combined with the assurance of machine-checked proof in a framework that is adequate for modern research-level mathematics. Related work that one hopes is leading in this direction includes work on constructing a hierarchy of algebraic structures in Coq using type classes [SvdW11] or canonical structures [Gar11] as well as work on infrastructure for analysis and geometry [HIH13, Har13].

Acknowledgments

My thanks are due to the referees for their constructive and helpful remarks.

References

[AJ 1] Rob Arthan and Roger Bishop Jones. Z in HOL in ProofPower. *BCS FACS FACTS*, 2005-1. `http://www.lemma-one.com/ProofPower/index/`.

[Art14] R.D. Arthan. Product mathematical case studies, 2004-2014. `http://www.lemma-one.com/ProofPower/examples/examples.html`.

[BLM12] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Improving real analysis in coq: A user-friendly approach to integrals and derivatives. In Chris Hawblitzel and Dale Miller, editors, *Certified Programs and Proofs - Second International Conference, CPP 2012, Kyoto, Japan, December 13-15, 2012. Proceedings*, volume 7679 of *Lecture Notes in Computer Science*, pages 289–304. Springer, 2012.

[CF02] Luis Cruz-Filipe. Formalizing real calculus in Coq. In Victor Carreño, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher*

Order Logics, Proceedings of the Workshop on Formalising Continuous Mathematics, Hampton, VA, USA, August 19, 2002, pages 158–166. NASA Technical Report NASA/CP-2002-211736, 2002.

[EH02] Martín Escardó and Reinhold Heckmann. Topologies on spaces of continuous functions. *Topol. Proc.*, 26(2):545–564, 2002.

[FL96] Roland Fettig and Bernd Löchner. Unification of higher-order patterns in a simply typed lambda-calculus with finite products and terminal type. In Harald Ganzinger, editor, *Rewriting Techniques and Applications, 7th International Conference, RTA-96, New Brunswick, NJ, USA, July 27-30, 1996, Proceedings*, volume 1103 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 1996.

[Fle00] Jacques D. Fleuriot. On the mechanization of real analysis in Isabelle/HOL. In Harrison and Aagaard [HA00], pages 145–161.

[Fre70] Peter Freyd. On the concreteness of certain categories. Sympos. Math., Roma 4, Teoria Numeri, Dic. 1968, e Algebra, Marzo 1969, 431-456 (1970)., 1970.

[Fre04] Peter Freyd. Homotopy is not concrete. *Repr. Theory Appl. Categ.*, 2004(6):1–10, 2004. `http://www.tac.mta.ca/tac/reprints/`.

[Gar11] François Garillot. *Generic Proof Tools and Finite Group Theory.* Theses, Ecole Polytechnique X, December 2011.

[Gor00] Mike Gordon. From LCF to HOL: a short history. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 169–186. The MIT Press, 2000.

[Got00] Hanne Gottliebsen. Transcendental functions and continuity checking in PVS. In Harrison and Aagaard [HA00], pages 198–215.

[HA00] John Harrison and Mark Aagaard, editors. *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science.* Berlin: Springer-Verlag, 2000.

[Har98] John Harrison. *Theorem Proving with the Real Numbers.* Berlin: Springer-Verlag, 1998.

[Har13] John Harrison. The HOL light theory of euclidean space. *J. Autom. Reasoning*, 50(2):173–190, 2013.

[HIH13] Johannes Hölzl, Fabian Immler, and Brian Huffman. Type classes and filters for mathematical analysis in isabelle/hol. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, volume 7998 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2013.

[Hod93] Wilfrid Hodges. *Model theory.* Cambridge: Cambridge University Press, 1993.

[Mac98] Saunders Mac Lane. *Categories for the Working Mathematician. 2nd ed.* New York, NY: Springer, 2nd ed edition, 1998.

[May99] J.P. May. *A Concise Course in Algebraic Topology.* Chicago, IL: University of Chicago Press, 1999.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.

[Nip93] Tobias Nipkow. Functional unification of higher-order patterns. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science, 19-23 June 1993, Montreal, Canada*, pages 64–74. IEEE Computer Society, 1993.

[Spa95] Edwin H. Spanier. *Algebraic Topology.* Berlin: Springer-Verlag, 1995.

[Spi92] J. Michael Spivey. *Z Notation - a reference manual (2. ed.).* Prentice Hall International Series in Computer Science. Prentice Hall, 1992.

[Sti09] Colin Stirling. Decidability of higher-order matching. *Logical Methods in Computer Science*, 5(3), 2009.

[SvdW11] Bas Spitters and Eelis van der Weegen. Type classes for mathematics in type theory. *Mathematical Structures in Computer Science*, 21(4):795–825, 2011.

[Tur79] D.A. Turner. Another algorithm for bracket abstraction. *J. Symb. Log.*, 44:267–271, 1979.