Edinburgh Research Explorer

Automating change of representation for proofs in discrete mathematics (extended version)

Citation for published version:
Raggi, D, Bundy, A, Grov, G & Pease, A 2016, 'Automating change of representation for proofs in discrete mathematics (extended version)' Mathematics in Computer Science, vol. 10, no. 4, pp. 429–457. DOI: 10.1007/s11786-016-0275-z

Digital Object Identifier (DOI):
10.1007/s11786-016-0275-z

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Mathematics in Computer Science

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 19. Feb. 2019

Automating change of representation for proofs in discrete mathematics (extended version)

Daniel Raggi[1*], Alan Bundy[1], Gudmund Grov[2], and Alison Pease[3]

[1] School of Informatics, University of Edinburgh
[2] School of Mathematical & Computer Sciences, Heriot-Watt University
[3] School of Computing, University of Dundee

Abstract. Representation determines how we can reason about a specific problem. Sometimes one representation helps us to find a proof more easily than others. Most current automated reasoning tools focus on reasoning within one representation. There is, therefore, a need for the development of better tools to mechanise and automate formal and logically sound changes of representation.

In this paper we look at examples of representational transformations in discrete mathematics, and show how we have used tools from Isabelle's Transfer package to automate the use of these transformations in proofs. We give an overview of a general theory of transformations that we consider appropriate for thinking about the matter, and we explain how it relates to the Transfer package.

We show a few reasoning tactics we developed in Isabelle to improve the use of transformations, including the automation of search in the space of representations. We present and analyse some results of the use of these tactics.

Keywords: change of representation, transformation, automated reasoning, Isabelle proof assistant

1 Introduction

Many mathematical proofs involve a change of representation from a domain in which it is difficult to reason about the entities in question to one in which some aspects essential to the proof become evident and the proof falls out naturally.

Many times the transformation makes it explicitly into the written proof, but sometimes it remains hidden as part of the involved process of coming up with the proof in the mathematician's mind. For a formal, mechanical proof, this can be problematic. First of all because the logical validity of the transformation needs to be accounted for, i.e., there must be a formal justification of it. Moreover, to advance towards the goal of automating mathematical reasoning, we must understand, in computational terms, the process of decision and application of appropriate transformations. Particularly, for a mechanical theorem

* This work has been supported by the Mexican Council of Science and Technology (CONACYT), with scholarship no. 214095.

prover to find a proof in the way that a mathematician would, it must be able to incorporate something like the hidden transformations going on inside the mathematician's mind.

We believe that the kind of representational changes we present in this paper is ubiquitous and extremely important in the practice of mathematics at all levels; from basic everyday maths, to advanced research maths. The importance of representational changes in mathematics is explicitly evidenced in historically notable works such as Kurt Gödel's incompleteness theorems, where the proof involves matching (or encoding) meta-theoretical concepts like 'sentence' and 'proof' as natural numbers, or more recently Andrew Wiles' proof of Fermat's Last Theorem, which involves matching the Galois representations of elliptic curves with modular forms. This phenomenon is also seen in refinement based formal methods (e.g. VDM and B): one starts with a highly abstract representation that is easy to reason with, and then it is step-wise refined to a very concrete representation that can be implemented as a computer program. All of these transformations are justified by a general notion of morphism.

The scientific hypothesis for which this paper provides evidence is:

> *Change of representation, via structural transformations, is a valuable tool for the construction of proofs. In the context of interactive computer mathematics, the value stems mainly from the reduction of effort required from the user, and the quality*[1] *of the proofs produced.*

In the course of providing evidence for this hypothesis, we present a few technical and theoretical contributions. They set the context in which the hypothesis is evaluated. Mainly:

- A general mathematical framework suitable for reasoning about representation changes between theories in higher-order logics. This is a theory of transformations concerning the semantics behind the mechanisms of Isabelle's *Transfer* package [11].
- A catalogue of transformations we have identified as important and useful for reasoning in discrete mathematics, along with their formalisation in Isabelle, and a set of mechanical proofs using these.
- The implementation of tools for Isabelle, designed to extend the tools of the Transfer package, mainly for automating the search in the space of representations, and the selection of useful ones.

In section 2 we give a brief introduction to Isabelle and the Transfer package. In section 3 we motivate the use of transformations with some examples of typical transformations used for reasoning in discrete mathematics. In section 4 we present the theory, and demonstrate how the tools of Isabelle's Transfer package fit into our notion of transformation. In section 5 we present the catalogue of transformations and briefly show how we have formalised them in Isabelle, for their use in proofs aided by the the tools of said package. In section 6 we present the design of a few reasoning tactics that we have implemented in Isabelle to

[1] A few measures of quality are discussed.

automate the search of suitable representations. In section 7 we present and analyse some results of their use in the light of the hypothesis stated above. In section 8 we conclude with a brief discussion of related work.

2 Background

Isabelle/HOL is a theorem proving framework based on a simple type-theoretical higher-order logic [18]. It is one of the most widely used proof assistants for the mechanisation of proofs. Apart from ensuring the correctness of proofs written in its formal language, Isabelle has powerful automatic tactics such as `simp` and `auto`, and through time it has been enriched with some internally-verified theorem provers such as `metis` [12] and `smt` [22], along with a connection from the internal provers to some very powerful external provers such as E, SPASS, Vampire, CVC3 and Z3 through the Sledgehammer tool [19]. Despite their power, none of these provers incorporate arbitrary transformations within their search for proofs, although the external ones necessarily require exactly one transformation from Isabelle/HOL into their own logic.

The Transfer package was first released for Isabelle 2013-1 as a general mechanism for defining quotient types and transferring knowledge from the old 'representation' type into the new 'abstract' type [11]. However, their generalisation is not restricted to the definition of new quotient types, but allows the user to relate constants between any two types by theorems of a specific shape called *transfer rules*. Some of these rules are automatically generated when the user defines a new quotient type, but the user is free to add them manually, provided that they prove a preservation theorem. Central to this package, the `transfer` and `transfer′` tactics try to automatically match the goal statement to a new one related by either equivalence (the former) or implication (the latter), inferring this relation from the transfer rules.

We have taken full advantage of the generality of the transfer package as a means of automating the translation between sentences across domains which are related by what we consider an appropriate and general notion of *structural transformation*. In Section 4 we give an overview of our notion of transformation and how the tactics of the transfer package are useful mechanisms for exploiting the knowledge of a structural transformation.

3 Motivation and vision

The worlds of mathematical entities are interconnected. Numbers can be represented as sets, pairs of sets, lists of digits, bags of primes, etc. Some representations are only *foundational* and the reasoner often finds it more useful to discard the representation for practical use (e.g., natural number 3 is represented by $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$ in the typical ZF foundations, but this representation is rarely used in practice), and some are *emergent*; they only come about after having accumulated a fair amount of knowledge about the objects themselves, but are more helpful as reasoning tools (e.g., natural numbers as bags of primes). Overall, we think that there is no obvious notion of 'better representation', and it is up to the reasoner to choose, depending on the task at hand. Thus, we envision a system where the representation of entities can be fluidly transformed.

We have looked at some problems in discrete mathematics and the transformations commonly used for solving them. Below, we give one motivating example and show how we have mechanised the transformation in question inside Isabelle/HOL. Other motivating examples are briefly mentioned.

Numbers as bags of primes

Let us start with an example of the role of representation in number theory. Consider the following problem:

Problem 1. Let n be a natural number. Assume that, for every prime p that divides n, its square p^2 also divides it. Prove that n is the product of a square and a cube.

A standard solution to this problem is to take a set of primes p_i such that $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$. Then we notice that the condition "if p divides n then p^2 also divides n" means that $a_i \neq 1$, for each a_i. Then, we need to find $x_1, x_2, \ldots, x_k$ and $y_1, y_2, \ldots, y_k$ where

$$(p_1^{x_1} p_2^{x_2} \cdots p_n^{x_k})^2 (p_1^{y_1} p_2^{y_2} \cdots p_n^{y_k})^3 = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_k}$$

or simply

$$2(x_1, x_2, \ldots, x_k) + 3(y_1, y_2, \ldots, y_k) = (a_1, a_2, \ldots, a_k).$$

Thus, we only need to prove that for every $a_i \neq 1$ there is a pair x_i, y_i such that $2x_i + 3y_i = a_i$. The proof of this is routine.

The kind of reasoning used for this problem is considered standard by mathematicians. However, it is not so simple in current systems for automated theorem proving. The non-standard step is the 'translation' from an expression containing various applications of the exponential function into a simpler form, that appears to be stated in something like a linear arithmetic of lists, validated by the fundamental theorem of arithmetic.

The informal nature of the argument, in the usual mathematical presentation, leaves it open whether the reasoning is best thought as happening in an arithmetic of lists where the elements are the exponents of the primes, or perhaps in a theory of bags (also called 'multisets'; through this paper we use the words interchangeably) where the elements are prime numbers, or perhaps through a double translation, first representing numbers as bags of primes, and then representing these as lists where the elements are the multiplicities in the order of their respective primes. The reader might find it very easy to fluidly understand how these representations match with each other and how they are really just different aspects of the same thing. Such ease supports our overall argument and vision: that to automate mathematical reasoning, we require a framework in which data structures are linked robustly by logically valid translations, where the translation from one to another is easily conjured up.

The *numbers-as-bags-of-primes* transformation that links each positive integer to the bag of its prime factors is valid because there are operations on each side (numbers and multisets) that correspond to one another. For example, 'divides' corresponds to 'sub(multi)set', 'least common multiple' corresponds to 'union', 'product' corresponds to 'multiset addition', etc. Furthermore, all the predicates used in the statement of problem 1 have correspondences with well-known predicates regarding bags of primes. Thus, the problem can be translated as a whole. Other representations may not be very productive, e.g., try thinking about exponentiation in terms of lists of digits.

Table 1 shows more examples of number theory problems with their corresponding problem about multisets.

Problem in $\mathbb{N}$	**Problem in multisets**
Prove that there exist $x, y, z \in \mathbb{N}$ where $\texttt{gcd}(x, y) \neq 1$, $\texttt{gcd}(x, z) \neq 1$ and $\texttt{gcd}(y, z) \neq 1$, but $\texttt{gcd}(x, y, z) = 1$.	Prove that there exist x, y, z multisets where $x \cap y \neq \{\}$, $x \cap z \neq \{\}$ and $y \cap z \neq \{\}$, but $x \cap y \cap z = \{\}$.
Prove that for any positive integer n there exists a set A where $\|A\| = n$ and x divides yz for any x, y, and z in A.	Prove that for any positive integer n there exists a set A where $\|A\| = n$ and $x \subseteq y \uplus z$ for any x, y, and z in A.

Table 1. Number theory problems and their multiset counterparts. In the lower right corner, $\uplus$ refers to the *multiset addition*, where the multiplicities are added per element.

Numbers as sets

Many numerical problems have combinatorial proofs. There are two overlapping proof techniques involved in this method. One called *double counting*, and the other called *bijective proof.* For the former, these are proofs where the cardinality of a set is 'counted' in two different ways, resulting in two different numerical expressions; which ought to be equal because they count the cardinality of the same set. For the latter, two sets are found with a bijection between them, showing that their sizes are equal.

In general, the informal notion of *counting* consists of applying a variety of results regarding the sizes of sets with respect to set operations. For example, if a set is partitioned and we know the size of all the parts, then we can infer the size of the original set. If a set is bijected with another set for which we know the size, then we know the size of the original. All of these are methods for counting. Then, the double counting method really consists of viewing sets in terms of different set operations in a manner such that both expressions are 'connected' with numerical expressions. The way in which these connections are inferred is an instance of the overall topic of this paper.

Table 2 shows examples of arithmetic problems with their corresponding finite set theory problems. While the proofs of the numerical versions are not obvious at all (some of which are important results in basic combinatorics), the proofs of their finite set versions can be considered routine.

Problem in $\mathbb{N}$	**Problem in sets**
$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$	The set $\{x \subseteq \{0,1,\ldots,n\} : \lvert x\rvert = k+1\}$ can be partitioned into 2 parts: those that contain element n and those that do not.
$\binom{n}{k} = \binom{n}{n-k}$	There is a bijection between the subsets of $\{1,\ldots,n\}$ with k elements and those with $n-k$ elements
$2^n = \sum_{i=0}^{n} \binom{n}{i}$	The power set of $\{1,\ldots,n\}$ can be partitioned into $n+1$ parts $X_0, X_1, \ldots, X_n$ where $\lvert x\rvert = i$ for every $x \in X_i$.

Table 2. Numerical problems and their set counterparts.

Interconnectedness

We want to stress the importance of having fluidity of representations. For example, we talked about the ease with which we could think that the *numbers-as-bags-of-primes* transformation is actually a transformation of numbers to a theory of lists, where elements of the list are the exponents of the ordered prime factors. Inspired by this, we have mechanised many other simple transformations, but whose composition allows us to translate fluently from one representation to another. Our global vision of transformations useful in discrete mathematics, which we have mechanised[2], is represented in Figure 1. It is worth mentioning that the diagram is not commutative and that it abstracts logical relations (information may be lost, so some paths can only be traversed in one direction).

In the next section we show how a notion of transformation that accounts for this kind of correspondence between structures can be applied in formal proofs using Isabelle's Transfer tool.

[2] These can be found in http://dream.inf.ed.ac.uk/projects/rerepresent/.

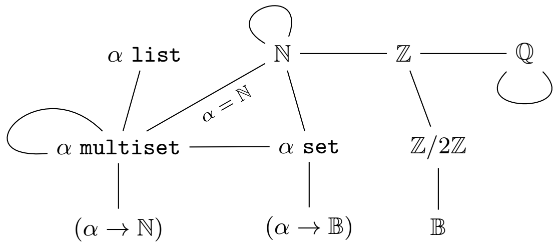

Fig. 1. Nodes represent structures (or corresponding theories of them), and edges represent the existence of at least one transformation connecting them. Apart from the aforementioned transformations, it includes other simpler ones. Some of these transformations are polymorphic (parametric on a type α), but others are not (like the one from $\mathbb{N}$ to $\mathbb{N}$ `multiset`, which relates natural numbers with multisets of primes, so it requires that $\alpha = \mathbb{N}$).
Node $\mathbb{Z}/2\mathbb{Z}$ stands for the structure of the *bit* type (constructed as a quotient of integers), and $\mathbb{B}$ stands for the boolean type.

4 On Transformations and the Transfer tool

In this section we introduce a theoretical framework for understanding structures and transformations. We tie together our own very general semantic notion *transformation* (a certain kind of morphism between higher-order type-theoretical models, that accounts for structure-preserving mappings, or morphisms in the algebraic sense) to the formalisms and mechanisms used in the *transfer* package [11] for Isabelle/HOL, as well as the formalisms of generalised rewriting [20] currently being implemented in the theorem prover Coq [23].

The theory we present in this paper is a semantic account of what the Transfer methods do, i.e., we investigate what the derivations made by the transfer methods mean in terms of transformations of the underlying structures. We believe that our approach brings some clarity to the problem at hand, and may help to strengthen the link between the algebraic notions of morphism and practical tools for theorem provers.

4.1 Structures

First we study in what kind of world the entities in question (mathematical objects) live, and then we see how a notion of transformation with certain desirable properties can fit into this world. The semantics we give to Isabelle/HOL theories is similar to that given in [14] for the HOL Light kernel with conservative extensions, and in [15] for Isabelle/HOL for slightly more complex extensions. Types are interpreted as non-empty sets, there is a distinguished set $\mathbb{B}$ which consists of boolean values $\{\top, \bot\}$, functions are interpreted as classical set-theoretical functions, the truth of statements depends on whether they map to element $\top$, and interpretations are parametrised over the type variables of polymorphic types[3]. We will see that transformations can be seen as relations between models

[3] Polymorphic types are type families over which functions can be defined without the need of specifying a concrete type.

of HOL theories and, more interestingly, that transformations themselves may be contained in extensions of the models.

We define a structure from the bottom up. In traditional model-theoretical formalisms, a universe (or superstructure) is defined by a class of basic entities (often just the empty set), and the rest of the structure is defined by the recursive (often transfinite) application of the power-set constructor function. In the context of higher-order logic we do it analogously, using the function constructor $(\to)$.

Definition 1. If T is a set of types, the *functional type structure* $T^{\to}$ is defined recursively as follows:

- if $\tau \in T$ then $\tau \in T^{\to}$
- if $\tau_1 \in T^{\to}$ and $\tau_2 \in T^{\to}$ then $\tau_1 \to \tau_2 \in T^{\to}$.

In other words, $T^{\to}$ is the closure of T over the constructor $\to$. Given that we are only interested in *functional* type structures, we can refer to them simply as *type structures.*

The type structure provides the frame of a structure, but the actual entities to which terms in HOL refer are the elements of these types. With the interpretation of types as sets and families of functions between them, we can define interpretations (models or superstructures) for theories in HOL, which helps us to reframe things in a familiar set-theoretic context.

Definition 2. We assume every type has a non-empty set associated with it. We call this its *universe.* The universe of a type τ is written $|\tau|$. If τ_1 and τ_2 are types, we define $|\tau_1 \to \tau_2|$ as the set of all functions from $|\tau_1|$ to $|\tau_2|$.

Then we can extend the notion of universe to sets of types, so that we can talk, for example, about the universe of a type structure $T^{\to}$.

Definition 3. If T is a set of types, we define its universe $\mathscr{U}_T$ as $\bigcup_{\tau \in T} |\tau|$. In other words, $\mathscr{U}_T$ is the set of entities with any type τ, with $\tau \in T$. Then, we can refer to $\mathscr{U}_{T^{\to}}$ as the *superstructure* of T; a universe that contains the entities associated to T, plus all the definable functions between them. We also refer to $\mathscr{U}_T$ as the *ground* of $\mathscr{U}_{T^{\to}}$.

Defining superstructures is not unmotivated. As we will see, the most common structures of traditional mathematics (sets, groups, rings, spaces, ...) are all captured inside superstructures. In the same manner, we will define a notion of transformation in a way that the respective notions of morphism (for the aforementioned structures) is accounted for.

Example 1. Let $\mathbb{N}$ be the natural number type and $\mathbb{B}$ be the boolean type. If $N = \{\mathbb{N}, \mathbb{B}\}$, its superstructure $\mathscr{U}_{N^{\to}}$ contains all natural numbers, plus the basic arithmetic operations (e.g., $\texttt{Suc}, +, *, \ldots$). Moreover, our favourite arithmetic relations also live there $(=, <)$ because they can be represented as boolean-valued functions (with type $\mathbb{N} \to \mathbb{N} \to \mathbb{B}$). Furthermore, there are logical operator-entities $\neg$, $\wedge$, $\vee$, $\longrightarrow$ as well as quantifier-entities $\forall, \exists$ (which have type $(\mathbb{N} \to \mathbb{B}) \to \mathbb{B}$). Thus, a full interpretation of a basic theory of arithmetic can be given.

In such a superstructure of natural numbers, we can find, amongst all possible operators, those giving it a monoid (a semigroup with identity) structure, those giving it a linear order structure, or even those giving it ad-hoc topologies (sets can be represented as functions from $\mathbb{N}$ to $\mathbb{B}$, and the set operators as functions with $\mathbb{N} \to \mathbb{B}$ as arguments and values).

In general, any theory in Isabelle/HOL which uses no type variables has a trivial interpretation into a superstructure. *Polymorphic* theories, which make use of type variables (e.g., where we have theorems about entities of $\alpha\,\texttt{set}$, for any type α), have an interpretation for each valid instantiation of the type variables. In other words, a polymorphic theory refers to many superstructures at once.

4.2 Transformations

As with superstructures, we define transformations from the ground up; in terms of a *ground transformation* and an extension of it into a *superstructural transformation.* The relation between a ground transformation and a superstructural transformation will be analogous to the relation between a ground $\mathscr{U}_T$ and its superstructure $\mathscr{U}_{T^{\to}}$.

Let $\mathscr{U}_{A^{\to}}$ and $\mathscr{U}_{B^{\to}}$ be superstructures.

Definition 4. A *ground transformation* between $\mathscr{U}_A$ and $\mathscr{U}_B$ is a set $\mathscr{R}$ where every $R \in \mathscr{R}$ is a relation $R : \alpha \to \beta \to \mathbb{B}$ between some $\alpha \in A$ and some $\beta \in B$.

Note that the relational nature of a transformation makes it possible to transform one entity to many other entities, even of various types (by different relations belonging to the transformation).

We have given a way to relate two unstructured universes. To relate the superstructures we need some notion of 'extension' of the transformation. So the question is: given a transformation (in this case a set of relations) between two grounds, is there a natural way of extending it to the structures built on top of them? Below we suggest a general notion which accounts for the transformations we are interested in.

Definition 5. A *structure relator* $\mathcal{S}$ is any function that takes two relations R_1 of type $\alpha_1 \to \beta_1 \to \mathbb{B}$ and R_2 of type $\alpha_2 \to \beta_2 \to \mathbb{B}$, in which the result of the application $(\mathcal{S}\,R_1\,R_2)$ has type $(\alpha_1 \to \alpha_2) \to (\beta_1 \to \beta_2) \to \mathbb{B}$.

If $\mathscr{R}$ is a ground transformation between $\mathscr{U}_A$ and $\mathscr{U}_B$, and two relations R_1 and R_2 (of $\mathscr{R}$) relate entities from the grounds, the element $\mathcal{S}\,R_1\,R_2$ relates functions of the superstructures $\mathscr{U}_{A^{\to}}$ and $\mathscr{U}_{B^{\to}}$. Thus, we can see any $\mathcal{S}$ as a specific rule for extending a ground transformation $\mathscr{R}$ to the superstructures.

Definition 6. Let $\mathscr{R}$ be a transformation between two grounds $\mathscr{U}_A$ and $\mathscr{U}_B$, and let $\mathcal{S}$ be a structure relator. The *$\mathcal{S}$-structural extension* of $\mathscr{R}$, written $\mathscr{R}^{\mathcal{S}}$, is defined recursively as follows:

- If $R \in \mathscr{R}$ then $R \in \mathscr{R}^{\mathcal{S}}$.
- If $R_1 \in \mathscr{R}^{\mathcal{S}}$ and $R_2 \in \mathscr{R}^{\mathcal{S}}$ then $(\mathcal{S}\, R_1\, R_2) \in \mathscr{R}^{\mathcal{S}}$.

Thus, if ground transformation $\mathscr{R}$ relates $\mathscr{U}_A$ and $\mathscr{U}_B$, its $\mathcal{S}$-structural extension relates their respective superstructures $\mathscr{U}_{A^{\rightarrow}}$ and $\mathscr{U}_{B^{\rightarrow}}$. We refer to a structural extension of a ground transformation as a *$\mathcal{S}$-superstructural transformation* or simply as a *$\mathcal{S}$-transformation*.

Transforming functions and relations.
We have established a very general way of talking about structural transformations, based on structure relators. In this section we connect the concepts of our theory with the operations built into the transfer tool. In particular, we focus on the structure relator $\Mapsto$, which is the basis for the transfer mechanisms. We show that it accounts for the well known notions of morphism.

Definition 7. The *standard functional extension* of two relations $R_1 : \alpha_1 \rightarrow \beta_1 \rightarrow \mathbb{B}$ and $R_2 : \alpha_2 \rightarrow \beta_2 \rightarrow \mathbb{B}$ (written $R_1 \Mapsto R_2$) is a relation that relates two functions $f : \alpha_1 \rightarrow \alpha_2$ and $g : \beta_1 \rightarrow \beta_2$ whenever they satisfy the following property:

$$\forall\, a : \alpha_1,\ \ \forall\, b : \beta_1,\ \ (R_1\, a\, b \ \longrightarrow\ R_2\, (f\, a)\, (g\, b))$$

The operator $\Mapsto$ is what we call the *standard function relator.* We write $(R_1 \Mapsto R_2)\, f\, g$ to say that f and g are related by $(R_1 \Mapsto R_2)$, and it means that the functions f and g map arguments related by R_1 to values related by R_2. Notice that if R_1 and R_2 happen to be functions, the property is actually: $\forall\, a : \alpha_1,\ \forall\, b : \beta_1,\ (R_1\, a = b \ \longrightarrow\ R_2\, (f\, a) = g\, b)$ which is clearly equivalent to $R_2 \circ f = g \circ R_1$, showing that it generalises the notion of structure-preserving mapping appropriately.

Furthermore, it can be shown that the expression $(R_1 \Mapsto \cdots \Mapsto R_n \Mapsto R)\ f\ g$ actually means

$$\forall\, a_i.\ \forall\, b_i.\ (R_1\, a_1\, b_1 \wedge \cdots \wedge R_n\, a_n\, b_n \ \longrightarrow\ R\, (f\, a_1\, \ldots\, a_n)\, (g\, b_1\, \ldots\, b_n))\,,$$

which corresponds intuitively to the legend ‘related arguments map to related values’.

As a graphical aid, see the ‘commutative diagram’ below, where the edges representing the relations can be traversed in any direction. The multi-arguments of the functions are represented in their traditional product form for simplicity.

$$\begin{tikzcd}
\alpha_1 \arrow[d, dashed, leftrightarrow] & \times \cdots \times & \alpha_n \arrow[rr, "f"] \arrow[d, dashed, leftrightarrow] \arrow[ll, phantom, "R_1 \cdots R_n", yshift=-3em] & & \alpha \arrow[d, dashed, leftrightarrow, "R"] \\
\beta_1 & \times \cdots \times & \beta_n \arrow[rr, "g"'] & & \beta
\end{tikzcd}$$

As definition 6 specifies, we write $\mathscr{R}^{\Mapsto}$ to denote the $\Mapsto$-structural extension of $\mathscr{R}$.

Thus, $\mathscr{R}^{\Mapsto}$ can be seen as a morphism between superstructures. These morphisms account for many of the usual algebraic morphism between structures (e.g., monoid, group, or ring morphisms). It can also be shown that any class of superstructures and their transformations forms a category, and that this category is related to well known categories. For example, let $\mathcal{C}'$ be the category of groups, and $\mathcal{C}$ the category of $\Mapsto$-superstructures with $\Mapsto$-structural transformations as morphisms. Then, there is a functor $F : \mathcal{C}' \to \mathcal{C}$ that injects $\mathcal{C}'$ into $\mathcal{C}$ (to each group assign the superstructure that has a type of the elements of the group at the base). The group operations will exist in the functional superstructures, and traditional group morphisms can be mapped to corresponding transformations. We will not go into details, but we will demonstrate the usefulness of superstructrual transformations.

Let us demonstrate how the informal numbers-as-bags-of-primes transformation can be formalised using these concepts. Let $N = \{\mathbb{N}, \mathbb{B}\}$. Then, as shown in example 1, $\mathscr{U}_{N^{\to}}$ is an interpretation of basic number theory. Let $M = \{\mathbb{N}\,\texttt{multiset}, \mathbb{B}\}$. Then, $\mathscr{U}_{M^{\to}}$ is an interpretation of basic multiset theory. To build a transformation between these two superstructures we start with relations $\texttt{BN} : \mathbb{N}\ \texttt{multiset} \to \mathbb{N} \to \mathbb{B}$, which relates every positive natural number with the multiset of its factorisation in primes. As we will see in the examples below, the ground transformation also needs to include boolean relations $\texttt{eq}$, $\texttt{imp}$ and $\texttt{revimp}$[4]. Thus, we consider the $\Mapsto$-structural extension of $\{\texttt{BN}, \texttt{eq}, \texttt{imp}, \texttt{revimp}\}$, and show how the well known operators of the superstructure of N map to well known operators of the superstructure of M.

Example 2. Let $* : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ be the usual multiplication and $\uplus : \alpha\ \texttt{multiset} \to \alpha\ \texttt{multiset} \to \alpha\ \texttt{multiset}$ the 'addition' of multisets (in which the multiplicities are added per element). Then we have $(\texttt{BN} \Mapsto \texttt{BN} \Mapsto \texttt{BN})\ \uplus\ *$. This is due to the law of exponents $p^a p^b = p^{a+b}$.

Furthermore, the standard function relator can also be used to express morphisms that preserve properties and relations. This is because we represent relations as boolean-valued functions. Then, to represent relations as part of the superstructure of a ground we require it to have boolean type $\mathbb{B}$, and to represent

[4] The expression $\texttt{imp}$ stands for implication and $\texttt{revimp}$ stands for 'reverse implication'

relational extensions of a transformation we require the transformation to have boolean relations (e.g., implication and equivalence).

For example, given n-ary relations $r : \alpha_1 \to \cdots \to \alpha_n \to \mathbb{B}$ and $s : \beta_1 \to \cdots \to \beta_n \to \mathbb{B}$ and the relations of a transformation $R_1 : \alpha_1 \to \beta_1 \to \mathbb{B}, \cdots, R_n : \alpha_n \to \beta_n \to \mathbb{B}$ and $S : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ we write $(R_1 \Mapsto \cdots \Mapsto R_n \Mapsto S)\ r\ s$ to express that r and s are logically related (by S).

In particular, S may be implication, which would mean that whenever r holds s will hold for related values.

Example 3. Let $\texttt{dvd} : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ be the relation such that $\texttt{dvd}\, n\, m$ whenever n divides m, and $\subseteq : \alpha\ \texttt{multiset} \to \alpha\ \texttt{multiset} \to \mathbb{B}$ the relation such that $a \subseteq b$ whenever the multiplicity of each element of a is lesser or equal to its multiplicity in b. Then, we have $(\texttt{BN} \Mapsto \texttt{BN} \Mapsto \texttt{eq})\ \subseteq\ \texttt{dvd}$, because $n\ \texttt{dvd}\ m$ if and only if every prime is contained at least as many times in the multiset-factorisation of m as it is in n.

Transforming logical operators

Regardless of the specific superstructure on which we are working, we can say few general things about equality, the quantifiers, and the propositional logical operators. These results are not novel. In fact, the majority of theorems mentioned here were built into the transfer package by its authors, so we will just give an overview.

For example, it is easily shown that *right-uniqueness*[5] implies preservation of equality to the right, i.e.,

$$\texttt{right-unique}\ R \longrightarrow (R \Mapsto R \Mapsto \texttt{imp})\ \texttt{eq}\ \texttt{eq}.$$

Similarly, *left-uniqueness* implies preservation of equality to the left, and *bi-uniqueness* preserves it in both directions.

Example 4. The prime factorisation of a natural number is unique and it characterises the number. Therefore, it is bi-unique and thus $(\texttt{BN} \Mapsto \texttt{BN} \Mapsto \texttt{eq})\ \texttt{eq}\ \texttt{eq}$.

The classical logical operators also relate in simple ways through structural transformations. For example, we have

$$(\texttt{eq} \Mapsto \texttt{eq} \Mapsto \texttt{eq})\ \texttt{and}\ \texttt{and}.$$

In fact, every statement $(\texttt{eq} \Mapsto \cdots \Mapsto \texttt{eq})\, f\, f$ is true for any f and any number of $\Mapsto$ used. This is unsurprising. The statement simply means that any function is undisturbed by replacing its arguments with equal ones. Then, given that logical equivalence is simply boolean equality we have the same result for `or`, `imp`, `revimp`, `eq` and `not`. The following theorems are slightly more interesting:

[5] A binary relation R is said to be *right-unique* if $R\,a\,b$ and $R\,a\,c$ implies $b = c$. In other words, when every element of the left has a unique element of the right. This is also called **univalent**.

- $(\texttt{imp} \mapsto\!\!\Rightarrow \texttt{imp} \mapsto\!\!\Rightarrow \texttt{imp})\ \texttt{and}\ \texttt{and}$
- $(\texttt{imp} \mapsto\!\!\Rightarrow \texttt{imp} \mapsto\!\!\Rightarrow \texttt{imp})\ \texttt{or}\ \texttt{or}$
- $(\texttt{revimp} \mapsto\!\!\Rightarrow \texttt{imp} \mapsto\!\!\Rightarrow \texttt{imp})\ \texttt{imp}\ \texttt{imp}$
- $(\texttt{imp} \mapsto\!\!\Rightarrow \texttt{revimp} \mapsto\!\!\Rightarrow \texttt{imp})\ \texttt{revimp}\ \texttt{revimp}$
- $(\texttt{revimp} \mapsto\!\!\Rightarrow \texttt{imp})\ \texttt{not}\ \texttt{not}$

Note the use of `revimp` (reverse implication) for the implications and negation. For implication, it simply means: $P \longrightarrow Q$ *implies* $P' \longrightarrow Q'$ *if* P' *implies* P *and* Q *implies* Q', and similarly for reverse implication. For negation it means: $\neg P$ *implies* $\neg P'$ *if* P' *implies* P.

Trivial theorems like these, regarding the mappings between logical operators under transformations, abound. A large number of them are built into the transfer package already. There are others that were not included and which are necessary for the transformations of our work. We have added these manually. We clarify their usefulness in section 4.4.

A quantifier of the universe of type τ is an entity of type $((\tau \to \mathbb{B}) \to \mathbb{B})$. The relevant question is how a transformation extends structurally to quantifiers, i.e., when does the relation $((R \mapsto\!\!\Rightarrow S_1) \mapsto\!\!\Rightarrow S_2)$ hold, if R is a relation between types A and B, and S_1 and S_2 are boolean relations?

We mentioned that uniqueness theorems have implications on how equalities of different types relate through transformations. Similarly totality properties have implications on how quantifiers can be transformed. For example, regarding the universal quantifier we have the two following theorems:

$$\begin{aligned} \texttt{right-total}\,R &\longrightarrow ((R \mapsto\!\!\Rightarrow \texttt{imp}) \mapsto\!\!\Rightarrow \texttt{imp})\ \forall\ \forall \\ \texttt{left-total}\,R &\longrightarrow ((R \mapsto\!\!\Rightarrow \texttt{imp}) \mapsto\!\!\Rightarrow \texttt{imp})\ \exists\ \exists \end{aligned}$$

For bounded quantifiers we also have results. For example `BN` is neither right-total nor left-total, so we cannot extend it to the usual quantifiers, but we still have the following:

Example 5. Let $\forall_{>0} : ((\mathbb{N} \to \mathbb{B}) \to \mathbb{B})$ be such that $\forall_{>0}\, x.\, P\, x$ is true if and only if $\forall\, x > 0.\, P\, x$, for every property P. Then we have $((\texttt{BN} \mapsto\!\!\Rightarrow \texttt{imp}) \mapsto\!\!\Rightarrow \texttt{imp})\ \forall\ \forall_{>0}$. This is because every natural number greater than 0 has a factorisation into primes.

4.3 Inverse transformations

Due to the relational nature of transformations, each of them has an inverse. Let $\texttt{flip} : (\alpha \to \beta \to \mathbb{B}) \to (\beta \to \alpha \to \mathbb{B})$ be the operator that flips the arguments of a relation, i.e., for any relation R we have $R\, a\, b = (\texttt{flip}\, R)\, b\, a$. Then, `flip` generates the inverse of every relation. The natural way to define the inverse of a transformation $\mathscr{R}^{\mathcal{S}}$ would be to apply `flip` to every one of its relations (let us call this $(\mathscr{R}^{\mathcal{S}})^{-1}$). So, is this a superstructural transformation? In other words, is there a pair $(\mathscr{R}', \mathcal{S}')$ such that ${\mathscr{R}'}^{\mathcal{S}'} = (\mathscr{R}^{\mathcal{S}})^{-1}$? The answer is yes, and we provide them below.

In general, the inverse of a transformation $\mathscr{R}^{\mathcal{S}}$ can be built by taking the ground transformation $\mathscr{R}^{-1}$, resulting from flipping all the relations of $\mathscr{R}$, and taking the relator $\mathcal{S}^{-1}$ defined as $\mathcal{S}^{-1}\,R_1\,R_2 = \texttt{flip}\,(\mathcal{S}\,R_1\,R_2)$, for every R_1 and R_2. Then, it is easy to see that the transformation $(\mathscr{R}^{-1})^{\mathcal{S}^{-1}}$ is simply that in which all the relations of $\mathscr{R}^{\mathcal{S}}$ have been flipped. Therefore, it equals $(\mathscr{R}^{\mathcal{S}})^{-1}$.

In particular, the inverse $(\mathscr{R}^{\Mapsto})^{-1}$ of any $\Mapsto$-transformation is simply $(\mathscr{R}^{-1})^{\Mapsto}$. This can be shown using the following lemma.

Lemma 1. Let R_1 and R_2 be binary relations. Then we have:

$$\texttt{flip}\,(R_1 \Mapsto R_2) = (\texttt{flip}\,R_1 \Mapsto \texttt{flip}\,R_2)$$

Proof. We have to prove that the relations have equal values on all arguments. Let f and g be functions. Then we have the following chain of equivalences:

$$\begin{aligned}(\texttt{flip}\,(R_1 \Mapsto R_2))\,f\,g &= (R_1 \Mapsto R_2)\,g\,f\\ &= \forall a\,b.\, R_1\,a\,b \longrightarrow R_2\,(g\,a)\,(f\,b)\\ &= \forall b\,a.\,(\texttt{flip}\,R_1)\,b\,a \longrightarrow (\texttt{flip}\,R_2)\,(f\,b)\,(g\,a)\\ &= (\texttt{flip}\,R_1 \Mapsto \texttt{flip}\,R_2)\,f\,g\end{aligned}$$

Thus we conclude the proof. □

From this lemma it follows that $(\mathscr{R}^{\Mapsto})^{-1} = (\mathscr{R}^{-1})^{\Mapsto}$, which can be easily proved by induction over the number of times $\Mapsto$ is applied.

Moreover, we have the following trivial result:

Lemma 2. If R is a symmetric relation then $\texttt{flip}\,R = R$.

Proof. It follows immediately from the definitions of symmetry and `flip`. □

Lemmas 1 and 2 help us to mechanically calculate inverse transformations. In particular, the former implies that, to calculate the inverse of a $\Mapsto$-transformation, we only have to calculate the inverse of the ground and then extend normally using $\Mapsto$. We come back to this in section 5.3.

4.4 Transforming problems and theorems

We showed how superstructures relate to each other by transformations, and gave a few examples of particular entities that relate to each other (e.g., in the numbers-as-bags-of-primes transformation the operator $\subseteq$ of multisets maps to operation `dvd` of natural numbers).

This section is concerned with how transformations can be exploited for practical theorem proving. We explain how the mechanisms of the transfer package correspond to derivations about underlying superstructures and transformations. Here we only aim to connect the semantics with the practical mechanisms, and not to provide a full description of the mechanisms. For a description of the mechanisms see [11].

A statement in HOL is considered true if its interpretation into a superstructure is the element $\top$ in $\mathbb{B}$. Thus, the truth of a statement '$Q\,y$' depends on whether Q yields $\top$ when applied to y. Suppose we know that $(R \Mapsto \texttt{eq})\ P\,Q$. This means that Q will yield the same values as P for arguments related by R. Thus, the truth value of $Q\,y$ is the same as the truth value of $P\,x$, provided that $R\,x\,y$ holds. This is represented by the following sequent:[6]

$$\frac{\mathscr{R}^{\Mapsto} \models (R \Mapsto \texttt{eq})\,P\,Q \qquad \mathscr{R}^{\Mapsto} \models R\,x\,y \qquad \mathscr{U}_{A^{\to}} \models P\,x}{\mathscr{U}_{B^{\to}} \models Q\,y}$$

This sequent shows us that we can obtain knowledge about a superstructure $\mathscr{U}_{B^{\to}}$ using only knowledge about another superstructure $\mathscr{U}_{A^{\to}}$ and a transformation $\mathscr{R}^{\Mapsto}$ between them. It is important to notice that if we have two HOL theories with models $\mathscr{U}_{A^{\to}}$ and $\mathscr{U}_{B^{\to}}$ respectively, then we can construct a HOL theory for a model that contains both superstructures and the transformation $\mathscr{R}^{\Mapsto}$. This means that both the superstructures and the transformations can be explored within Isabelle/HOL, so we do not need an external theory to validate the application of transformations in mechanical proofs.

The sequent above is generalised by the elimination rule for the relator $\Mapsto$:

$$\frac{(R_1 \Mapsto R_2)\,f\,g \qquad R_1\,x\,y}{R_2\,(f\,x)\,(g\,y)} \tag{1}$$

This now concerns derivability, for any theory where the relator $\Mapsto$ is defined correctly. From this we can produce simple derivations such as the following:

$$\frac{\dfrac{(\texttt{BN} \Mapsto \texttt{BN} \Mapsto \texttt{imp})\ \subseteq\ \texttt{div} \qquad \texttt{BN}\,\{2,3\}\,6}{(\texttt{BN} \Mapsto \texttt{imp})\ (\lambda x.\,\{2,3\} \subseteq x)\ (\lambda x.\,6\,\texttt{div}\,x)} \qquad \texttt{BN}\,\{2,2,2,3,5\}\,120}{\texttt{imp}\ (\{2,3\} \subseteq \{2,2,2,3,5\})\ (6\ \texttt{dvd}\ 120)}$$

which shows that, given the appropriate knowledge about the numbers-as-bags-of-primes transformation, we can show 6 `dvd` 120 by proving that the bag of prime factors of 6 is contained in the bag of prime factors of 120 (we only give this example to illustrate the mechanism, not that this is a desirable way to prove that 6 divides 120).

Other derivations require to prove that two functions are related by a transformation (where it is not known in advance). This can be done using the introduction rule for the relator $\Mapsto$:

$$\frac{\forall\,x\,y.\ R_1\,x\,y \ \longrightarrow\ R_2\,(f\,x)\,(g\,y)}{(R_1 \Mapsto R_2)\,f\,g} \tag{2}$$

This rule simply means that, to derive that two functions are related, it suffices to assume $R_1\,x\,y$ for some fresh variables x and y, and show $R_2\,(f\,x)\,(g\,y)$.

[6] We overload the symbol $\models$ to say both that a structure satisfies a sentence relating to its theory, and that a superstructural transformation $\mathscr{R}^{\Mapsto}$ 'satisfies' a relation between two entities, one of each superstructure.

The `transfer` methods of Isabelle use rules 1 and 2 to build a derivation tree with an implication or equivalence as the conclusion. The leaves of the tree will all be statements regarding a transformation $\mathscr{R}^{\Mapsto}$ (either of shape $R\,x\,y$ or $(R_1 \Mapsto R_2)\,f\,g$). Statements about the transformation which are known to be true are called *transfer rules*, and they are distinguished in Isabelle by the attribute (a label) `transfer_rule`. If all the leaves of a derivation tree are transfer rules then we have obtained a statement (about the superstructure on the other side of the transformation) which implies our goal statement.

It is important to note that the relational nature of structural transformations implies that their application for inference, as implemented by the transfer methods, is non-deterministic. This is because there may be many transfer rules for the same constant, thus allowing the construction of various different derivation trees such as the one shown above.

We have briefly explained how knowledge about transformations can be used to make inferences in Isabelle. The `transfer` methods are tactics designed for making this kind of inference. Next we will present our applications of these methods.

5 Mechanising transformations in Isabelle/HOL

In Section 3 we presented some problems in discrete mathematics which involve structural transformation. We have mechanised the transformations by proving the necessary transfer rules. The transfer tool allows us to use the transformations in proofs.

In this section we present some examples from a larger catalogue of the transformations we have mechanised in Isabelle. Figure 1 shows a graph representing the connections of mathematical structures related to discrete mathematics. We have formalised plenty of transformations represented along the edges of the graph, and we have identified problems of discrete mathematics whose solutions rely on these transformations. However, the examples in which we have actually mechanised applications of the transformations to proofs we consider interesting for the field of discrete mathematics are restricted to only a subset of them; namely, items 1, 2, 3, 5, 6 and 8 of the list below, although others, such as, 7 are used in the proofs of transfer rules of other transformations (evidencing their usefulness). We have identified the rest to be potentially useful, but we have not performed interesting experiments with them.

1. **numbers-as-bags-of-primes**, where each natural number is related to the multiset of its prime factors.
2. **numbers-as-sets**, where numbers are related to sets by the cardinality function.
3. **sets-as-$\mathbb{B}$-functions**, where sets are seen as boolean-valued functions.
4. **set-to-multiset**, where every multiset is related to the set of its elements, ignoring multiplicity.

5. **set-in-multiset**, where finite sets are injected into multisets (the difference between this and transformation 4 is that this one is bi-unique; only multisets with multiplicities 0 and 1 are related).
6. **parametric multiset auto-transformations**, where multisets with one base type are related to multisets with another base type (not necessarily different), through transformations between the base types; these are parametric on the base transformation, but there are general things that can be said about them without the need to specify the base transformation. For this work, we have used the one-to-one mapping between natural numbers and prime numbers as the base transformation.
7. **multisets-as-lists**, where multisets are related to lists of their elements.
8. **multisets-as-ℕ-functions**, where multisets are seen as natural-valued functions.[7]
9. **naturals-as-integers**, where naturals are matched to integers (this one was built by the developers of the transfer package, not us).
10. **bits-from-integers**, where type `bit` is created as an abstract type from the integers.[8]
11. **bits-as-booleans**, where bits are matched to booleans.
12. **integers-as-rationals**, where integers are matched to rational numbers. Notice that composition of transformations leads to other natural transformations, such as the simple relation between sets and multisets.
13. **parametric rational auto-transformations**, where rational numbers are stretched and contracted, parametric on a factor.

Every transformation starts with a declaration and proof of transfer rules.

5.1 Numbers as bags of primes

The relation at the centre of this transformation is $\texttt{BN} : \mathbb{N}\ \texttt{multiset} \to \mathbb{N} \to \mathbb{B}$, which relates every positive number to the multiset of its prime factors. It is

[7] This one is actually by construction using `typedef` and the Lifting package, which automatically declares transfer rules from definitions lifted by the user from an old type to the newly declared type. This transformation was built by the authors of the Multiset theory in the library, when defining the multiset type. We added considerably to it.

[8] It is interesting to note that for every quotient $\mathbb{Z}/n\mathbb{Z}$ there is a transformation from $\mathbb{Z}$ which preserves the ring structure. These are extremely useful in discrete mathematics. Moreover, it is perfectly possible to define parametric transformations (e.g., with n as a parameter). However, it is not possible to define parametric types in Isabelle/HOL. Then, each $\mathbb{Z}/n\mathbb{Z}$ can be defined as a type manually, so ultimately only a finite number of them can be defined. Parametric types are definable in logics with richer type theories, such as Coq. The alternatives for us are to either build a finite number of them (which is obviously not ideal), or to define every $\mathbb{Z}/n\mathbb{Z}$ as a subset of the type $\mathbb{Z}$, with a structure of its own. This approach brings its own problems (e.g., that functions between them would need to be defined over the whole $\mathbb{Z}$, because Isabelle/HOL does not admit partial functions).

defined as follows: $\texttt{BN}\, M\, n$ holds if and only if

$$(\forall\, x \in M.\ \texttt{prime}\, x) \;\wedge\; n = \prod_{x \in M} x^{\texttt{count}\, M\, x}$$

where $\texttt{count}\, M\, x$ is the multiplicity of x in M; i.e., how many times x appears in M. The most basic transfer rules (instances of the structural transformation) are theorems such as $\texttt{BN}\ \{2,3\}\ 6$, whose proof are trivial calculations. Moreover, from the Unique Prime Factorisation[9] theorem we know that BN is bi-unique. Thus, we know that

$$(\texttt{BN} \Mapsto \texttt{BN} \Mapsto \texttt{eq})\, \texttt{eq}\, \texttt{eq}$$

i.e., that equality is preserved by the transformation.

From the fact that every positive number has a factorisation we have

$$((\texttt{BN} \Mapsto \texttt{imp}) \Mapsto \texttt{imp})\ \forall\ \forall_{>0} \qquad ((\texttt{BN} \Mapsto \texttt{imp}) \Mapsto \texttt{imp})\ \exists_p\ \exists$$
$$((\texttt{BN} \Mapsto \texttt{eq}) \Mapsto \texttt{eq})\ \forall_p\ \forall_{>0} \qquad ((\texttt{BN} \Mapsto \texttt{eq}) \Mapsto \texttt{eq})\ \exists_p\ \exists_{>0}$$

where imp is implication, $\forall_p$ is the bounded quantifier representing 'for every multiset where all its elements are primes' and $\forall_{>0}$ is the bounded quantifier representing 'for every positive number', and similarly for $\exists_p$ and $\exists_{>0}$. The mechanised proofs of these sentences follow in a relatively straightforward manner from the Unique Prime Factorisation theorem which is already part of Isabelle's library of number theory. To exemplify the meaning of these theorems, consider the statement in the top right. Informally, it means that if there exists a bag of primes satisfying a property P then there exists a number satisfying a similar property P', where P and P' are related by the transformation based on BN (specifically by $(\texttt{BN} \Mapsto \texttt{imp})$, which itself means that whenever P is satisfied by a bag of primes then P' is satisfied by its corresponding number).

Furthermore, we proved the following correspondences of structure:

$$(\texttt{BN} \Mapsto \texttt{BN} \Mapsto \texttt{BN})\ \uplus\ * \qquad (\texttt{BN} \Mapsto \texttt{BN} \Mapsto \texttt{BN})\ \cup\ \texttt{lcm}$$
$$(\texttt{BN} \Mapsto \texttt{BN} \Mapsto \texttt{BN})\ \cap\ \texttt{gcd} \qquad (\texttt{BN} \Mapsto \texttt{BN} \Mapsto \texttt{eq})\ \subseteq\ \texttt{dvd}$$
$$(\texttt{BN} \Mapsto \texttt{eq} \Mapsto \texttt{BN})\ \texttt{smult}\ \texttt{exp} \qquad (\texttt{BN} \Mapsto \texttt{eq})\ \texttt{sing}\ \texttt{prime}$$

where smult is the *scalar multiplication* of multisets, i.e., the multiplicites are multiplied by a scalar, and sing is the property of being a *singleton* multiset, i.e., a multiset of size 1.

Application in proofs

We formalised the proof of problem 1:

> *Let n be a natural number. Assume that, for every prime p that divides n, its square p^2 also divides it. Prove that n is the product of a square and a cube.*

[9] The Unique Prime Factorisation has a proof in the library of Isabelle-2015

First, notice that the case for $n = 0$ is trivial. Then, we state it formally for $n > 0$ (the automation of this kind of case splitting is addressed in section 6):

$$\forall\, n > 0.\ (\forall\, p.\ \texttt{prime}\ p\ \wedge\ p\ \texttt{dvd}\ n\ \longrightarrow\ p^2\ \texttt{dvd}\ n)\ \longrightarrow\ (\exists\, a\ b.\ a^2 * b^3 = n)$$

When we apply the transfer method to the sentence we get the following sentence about multisets:

$$\forall_p\, n.\ (\forall_p\, p.\ \texttt{sing}\ p\ \wedge\ p \subseteq n\ \longrightarrow\ 2 \cdot p \subseteq n)\ \longrightarrow\ (\exists_p\, a\ b.\ 2 \cdot a + 3 \cdot b = n)$$

where $\forall_p$ is the universal quantifier bounded to multisets of prime numbers, and operator $\cdot$ represents the scalar multiplication previously referred to as `smult` (we present it as we do for reading ease).

The premise $(\forall_p\, p.\ \texttt{sing}\ p\ \wedge\ p \subseteq n\ \longrightarrow\ 2 \cdot p \subseteq n)$ is easily proved to be equivalent to $\forall\, q.\ \texttt{count}\, n\, q \neq 1$. Then it is sufficient to show

$$\forall_p\, n.\ (\forall\, q.\ \texttt{count}\, n\, q \neq 1)\ \longrightarrow\ (\exists_p\, a\ b.\ 2 \cdot a + 3 \cdot b = n)$$

With some human interaction, this can be further reduced to proving that, for every element of n, its multiplicity n_i can be written as $2a_i + 3b_i$ (knowing that $n_i \neq 1$). Formally, this is stated as:

$$\forall\, n_i : \mathbb{N}.\ n_i \neq 1\ \longrightarrow\ \exists\, a_i\ b_i.\ 2a_i + 3b_i = n_i$$

This problem is expressible in a decidable part of number theory (Presburger arithmetic), for which there is a method implemented in Isabelle. We mechanised the proof using this method.

Notice that a problem, originally stated in terms of exponentiation and multiplication, is in the end reduced to one about multiplication and addition. This characterises the usefulness of the numbers-as-bags-of-primes transformation. After applying the transformation, we do some reasoning regarding the multiplicities of the elements (which are themselves natural numbers). In fact, it should be noted that in the actual mechanical proof we used more than one transformation (namely, transformations 6 and 8, as enumerated in the beginning of this chapter, as well as the numbers-as-bags-of-primes transformation that we are exemplifying here).

5.2 Numbers as sets

At the centre of this transformation is the relation `SN` where $\texttt{SN}\, A\, n$ holds if and only if $\texttt{finite}\, A \wedge |A| = n$.[10]

[10] The condition to be finite is needed because the cardinality function, in Isabelle, is defined to be 0 for infinite sets. This exemplifies the kind of concessions that need to be made in type-theory based systems like Isabelle. The alternatives are to introduce an intermediate type of finite sets or an intermediate type of transfinite cardinalities and work on the transformations linking them. This shows yet another motivation to focus on transformations.

We first prove trivial cardinality properties such as, $\mathtt{SN}\,\{1\cdots n\}\,n$, which allows us to consider standard representatives of numbers.

This relation is right-total but not left-total, so we have the following two rules:

$$((\mathtt{SN} \Mapsto \mathtt{imp}) \Mapsto \mathtt{imp})\ \forall\ \forall \qquad\qquad ((\mathtt{SN} \Mapsto \mathtt{eq}) \Mapsto \mathtt{eq})\ \forall_{\mathtt{fin}}\ \forall$$

where $\forall_{\mathtt{fin}}$ is the universal quantifier restricted to finite sets. Furthermore, the relation is left-unique but not right-unique, so we have

$$(\mathtt{SN} \Mapsto \mathtt{SN} \Mapsto \mathtt{imp})\ \mathtt{eq}\ \mathtt{eq} \qquad\qquad (\mathtt{SN} \Mapsto \mathtt{SN} \Mapsto \mathtt{eq})\ \mathtt{eqp}\ \mathtt{eq}$$

where eqp is the relation of being equipotent, or bijectable.

Then, we have the following rules for the structural correspondence:

$$\begin{aligned}(\mathtt{SN} \Mapsto \mathtt{SN})\ &\mathtt{Pow}\ (\lambda x.\, 2^x)\\ (\mathtt{SN} \Mapsto \mathtt{eq} \Mapsto \mathtt{SN})\ &\mathtt{nPow}\ \left(\lambda\, n\, m.\, \tbinom{n}{m}\right)\\ (\mathtt{SN} \Mapsto \mathtt{SN} \Mapsto \mathtt{imp})\ &\subseteq\ \leq\\ (\mathtt{SN} \Mapsto \mathtt{SN} \Mapsto \mathtt{SN} \Mapsto \mathtt{imp})\ &\mathtt{disjU}\ \mathtt{plus}\end{aligned}$$

where $\mathtt{nPow}\ S\ n$ returns the set of subsets of S that have cardinality n (we call these n-subsets). Also, $\mathtt{disjU}\ a\ b\ c$ means $\mathtt{disjoint}\ a\ b \wedge a \cup b = c$ and plus is the predicative form of operator $+$.

We have mechanised combinatorial proofs, such as the ones for the problems given in Table 2, of theorems using this transformation. Some of these are addressed in section 7.

5.3 Calculating inverse transformations

Apart from all the individual transformations we formalised, we developed a method for generating the inverse of any given transformation. This extends our inference capabilities given some knowledge about a superstructural transformation. For example, it allows us to logically reduce $P\,x$ to $Q\,y$ using the facts $R\,x\,y$ and $(R \Mapsto \mathtt{eq})\,P\,Q$ (rather than the other way around, as the transfer tactic does)[11]. The transfer tactic will only match constants of the goal from the right of a transfer rule, even if there is a sound derivation to be made by matching constants from the left of the transfer rule. In order to make inferences the other way around, we need a way of constructing *symmetric transfer rules*. We will show that any transfer rule always has a symmetric version, and how to obtain it mechanically (with a logically sound method). We will present the tool mk_symmetric_trule that does this; it takes a transfer rule as input and yields its symmetric version.

[11] Note that reducing $P\,x$ to $Q\,y$ cannot be done with $(R \Mapsto \mathtt{imp})\,P\,Q$, because of the direction of the implication, but it can be done with either $(R \Mapsto \mathtt{eq})\,P\,Q$, or with $(R \Mapsto \mathtt{revimp})\,P\,Q$.

At its essence, our conversion tool, `symmetric_trule`, is a systematic application of a set of rewrite rules concerning the behaviour of the operator `flip`, shown in section 4.3. From the definition of `flip` and lemma 1 we have the following equations:

$$R\,a\,b = (\texttt{flip}\,R)\,b\,a \tag{3}$$

$$\texttt{flip}\,(R_1 \Mapsto R_2) = (\texttt{flip}\,R_1 \Mapsto \texttt{flip}\,R_2) \tag{4}$$

Furthermore, from lemma 2 and the symmetry of equality we have:

$$\texttt{flip eq} = \texttt{eq} \tag{5}$$

Then, if we have a transfer rule $(R \Mapsto \texttt{eq})\,f\,g$, we can apply rules (3) and (4) to obtain $(\texttt{flip}\ R \Mapsto \texttt{flip eq})\,g\,f$. Finally, from (5) we can obtain the usable transfer rule $(\texttt{flip}\ R \Mapsto \texttt{eq})\,g\,f$. As we have shown before, transfer rules with equality (such as this one) are the kind that we can use to infer equivalence between a statement and its transformed version.

Furthermore, from the definition of `flip` we have the following facts, regarding implication and reverse implication:

$$\texttt{flip imp} = \texttt{revimp} \tag{6}$$

$$\texttt{flip revimp} = \texttt{imp} \tag{7}$$

Thus, for any transfer rule of the form $R\,a\,b$ (where R may be of the form $(R_1 \Mapsto R_2)$) we can rewrite it starting with rule (3), then recursively applying rule (4) (pushing `flip` in), and finish eliminating `flip` where possible, using rules (5), (6) and (7). This is exactly what our main conversion function, `mk_symmetric_trule`, does.

From this conversion function we build the Isabelle attribute `symmetric_trule`. Using it, a whole set of transfer rules can be reversed at once. A typical declaration of an entirely new set of symmetric transfer rules looks as follows:

`theorems NB_trules = BN_trules[symmetric_trule flip_intro[where R = BN]]`

Where `flip_intro[where R = BN]` is an instantiation of rule (3). If `BN_trules` is the set of transfer rules for transforming numbers into bags of primes, `NB_trules` will be the one for transforming bags into numbers, when possible.

6 Automated change of representation

Up to this point we have shown how discrete mathematics involves reasoning about a few superstructures which are heavily interconnected by transformations. We believe that to take full advantage of the connections provided we need some mechanisms for the automation of search between representations.

In this section we present the challenges and implementation of some tools necessary for the execution of automatic search in the space of representations.

Specifically, we present the implementation of the following tactics, written in Isabelle/ML:

`rerepresent_tac`: Tactic that processes sentences before and after applying Isabelle's transfer_tac.

`representation_search`: Basic tool for searching the space of representations, with atomic transformations handled by `rerepresent_tac`.

This section ought to be understood as presenting a bag of tools necessary for search, rather than a presentation of one specific use of search or the results of experimenting with it. The results of the experiments with search are presented in section 7.

6.1 Transformation knowledge as sets of transfer rules

As described in section 4, we consider a transformation as a set of ground relations, and a structural extension of them. Our knowledge about such a transformation can be seen as a collection of statements satisfied by the transformation, called transfer rules.

In the context of Isabelle, we simply package theorems into sets. A typical declaration of one of these sets looks as follows:

`theorems BN_trules =` BN_prod BN_set BN_factorization
BN_multiplicity BN_gcd . . . ,

where every argument is a named theorem; specifically, one that tells us how two operators match via a transformation. For example, the theorem `BN_prod` is $(\mathtt{BN} \Mapsto \mathtt{BN} \Mapsto \mathtt{BN}) \uplus *$.

When we want to apply a specific transformation, we just need to *turn on* the desired set of transfer rules and apply the transfer method.

6.2 Transformation preprocessing

There are two principal reasons why we would like to preprocess sentences before transforming. One is simply that, for some transformations, the transfer method requires sentences to be in a specific shape. In that case, the job of our preprocessing tools is simply to give the sentences that shape. The other reason is that some transformations require some semantic conditions to be met and which are not met by default (e.g., only positive natural numbers have prime factorisations, which means that many proofs have to be split in two cases: one for zero, which is often trivial, and one for the rest). For this matter, our preprocessing tools have to do a case split of the goal into two subgoals; one of which cannot be transformed but is trivial to prove, and one which is not trivial to prove but *can* be transformed.

Below we explain how our preprocessing tools work.

Normalising to transformation-specific language

We have built a tactic `tnormalise` which, if possible, normalises a goal to a language in which a transformation can be applied. The tactic is simple, but below we explain the motivation for building it.

The mechanism of Isabelle's transfer method matches atomic constants which appear in transfer rules. Generally, in an expression $f\,(g\,n)$, the constants f, g and n will have to match, via transfer rules, to constants in the target theory. This means that, even if we had a transfer rule $R\,h\,(\lambda\,n.\,f\,(g\,n))$, the possibility of matching $f\,(g\,n)$ to an expression of the form $h\,m$ would not be considered, the only reason being that $(\lambda\,n.\,f\,(g\,n))$ is not expressed with a single constant symbol. In our case, we often want to match complex operators, such as $\forall_{>0} : (\mathbb{N} \to \mathbb{B}) \to \mathbb{B}$ and $\forall_p : (\mathbb{N}\,\texttt{multiset} \to \mathbb{B}) \to \mathbb{B}$ (where $\forall_p$ expresses 'for all bags of primes'). However, quantifiers like these are unusual and inelegant in typical mathematical text; instead, we see them written as $\forall x > 0.\,Px$, or as $\forall x.\,x > 0 \longrightarrow P\,x$. Thus, before applying a transformation to a sentence, we sometimes need to fold (rewrite) definitions into specialised single-symbol constants.

Moreover, the need for normalising becomes evident when individual transformations are applied in the context of a broader search (the output statement of a transformation may not have the shape another transformation expects, so without user interaction or automatic normalisation the second transformation cannot be applied).

For each transformation we assign a set of definitions (rewrite rules) that we call *pretransformation definitions*, which are applied wherever it is possible, recursively. A typical declaration of these rules, in Isabelle, looks as follows:

```
theorems BN_pretrules = forall_nats_gr0_def exists_nats_gr0_def,
```

where `forall_nats_gr0_def` and `exists_nats_gr0_def` are the respective Isabelle-level representations of operators $\forall_{>0}$ and $\exists_{>0}$.

Moreover, a goal statement might not contain a pattern that directly matches a definition. For this reason, we include the possibility of adding extra elimination rules[12] associated with a definition. For example, we can add an extra (weaker, but true) rule such the following:

$$\texttt{forall\_nats\_gr0}\,P \longrightarrow (\forall\,n.\,0 < n \longrightarrow P\,n)$$

On the whole, the computation of function `tnormalise` consists of first resolving the goal statement with a set of elimination rules (determined by the transformation), and then *folding* a set of definition (also determined by the transformation).

[12] An elimination rule, used for backward reasoning, actually introduces the constant in question.

Case splitting

We have built a function `split_for_transfer` that takes a goal and returns two subgoals, one regarding the part of the universe where the transformation applies, and the other regarding the part where it does not apply. This allows us to split a goal and apply a transformation to the fraction where it is applicable; the rest will often be trivial to prove without a transformation.

Let us consider problem 1 again:

$$\forall n.\ (\forall p.\ \texttt{prime}\ p \wedge p\ \texttt{dvd}\ n \longrightarrow p^2\ \texttt{dvd}\ n) \longrightarrow (\exists\, a\, b.\ a^2 * b^3 = n)$$

In section 5.1 we showed the proof of this theorem for $n > 0$. The statement is true for all n, but it requires a separate proof for the case where $n = 0$. This case is actually trivial, but it blocks the transformation from being applied directly. Doing the case-split manually works if we know which transformation we want to apply and we only want to apply it once. However, if we want to automate the process for either browsing the space of representations or searching the space in a completely automated way, we need these case splits to be part of the preprocessing of every transformation.

For transformations which are partial on one of its types, we have assigned a case-splitting tactic, which generates two subgoals, one of which can be transformed, and another one which cannot.

Case-splitting, as a tactic, is an application of the theorem

$$((P \longrightarrow Q) \wedge (\neg P \longrightarrow Q)) \longrightarrow Q, \tag{8}$$

Then, given Q as a goal statement, we can reduce it to subgoals $(P \longrightarrow Q)$ and $(\neg P \longrightarrow Q)$. In particular, for the purpose of applying a transformation, we want P to be a proposition restricting the domain to which Q applies, so that statement $(P \longrightarrow Q)$ can be transformed, while handing subgoal $(\neg P \longrightarrow Q)$ to an automatic reasoning tactic, which may solve them or fail and leave them open.

For each transformation the user manually specifies a predicate representing a range in which the transformation applies. Let `P` be predicate of type $\beta \to \mathbb{B}$ and `st` a goal state. The application `split_for_transfer P st` splits `st` recursively for every universally-quantified variable of type β that appears in `st`. If the set of universally-quantified variables is $\{\mathtt{x}_0, \ldots, \mathtt{x}_\mathtt{n}\}$, it will yield a proposition (essentially) of the form

$$(((\mathtt{P}\,\mathtt{x}_0 \wedge \cdots \wedge \mathtt{P}\,\mathtt{x}_\mathtt{n}) \longrightarrow Q) \wedge (\neg(\mathtt{P}\,\mathtt{x}_0 \wedge \cdots \wedge \mathtt{P}\,\mathtt{x}_\mathtt{n}) \longrightarrow Q)) \longrightarrow Q.$$

Without loss of generality, assume our goal is $\forall\, \mathtt{x}_\mathtt{i} \in \{\mathtt{x}_0, \ldots, \mathtt{x}_\mathtt{n}\}.\ \mathtt{G}\,\mathtt{x}_0 \cdots \mathtt{x}_\mathtt{n}$. Then, substituting Q for $\mathtt{G}\,\mathtt{x}_0 \cdots \mathtt{x}_\mathtt{n}$ and resolving yields two subgoals:

$$\forall\, \mathtt{x}_\mathtt{i} \in \{\mathtt{x}_0, \ldots, \mathtt{x}_\mathtt{n}\}.\ (\mathtt{P}\,\mathtt{x}_0 \wedge \cdots \wedge \mathtt{P}\,\mathtt{x}_\mathtt{n}) \longrightarrow \mathtt{G}\,\mathtt{x}_0 \cdots \mathtt{x}_\mathtt{n} \tag{9}$$

$$\forall\, \mathtt{x}_\mathtt{i} \in \{\mathtt{x}_0, \ldots, \mathtt{x}_\mathtt{n}\}.\ \neg(\mathtt{P}\,\mathtt{x}_0 \wedge \cdots \wedge \mathtt{P}\,\mathtt{x}_\mathtt{n}) \longrightarrow \mathtt{G}\,\mathtt{x}_0 \cdots \mathtt{x}_\mathtt{n}. \tag{10}$$

Furthermore, subgoal (10) is handed over to the tactic `auto`, and subgoal (9) can be given to the `tnormalise` tactic, which can fold patterns such as '$\forall\, \mathtt{x}_\mathtt{i} \in \{\mathtt{x}_0, \ldots, \mathtt{x}_\mathtt{n}\}. \cdots$' to specific constants representing bounded quantifiers; the result of which can be handed over to a transformation-applying tactic.

6.3 Transformation postprocessing

The previous section explains how a goal has to be modified prior to applying a transformation. Similarly, we can modify a goal after a transformation to help with further reasoning. There are a couple of reasons for this. First, the language after a transformation may have symbols such as $\forall_p$, which may seem unusual, inelegant, or unnecessary to the user, and it can also prevent a successive transformation to be applied. Secondly, and more importantly, some transformations generate logically stronger subgoals, some of which may turn out to be false. Then, as part of the postprocessing, we can check for counterexamples and discard the transformation if it generates a provably false statement.

Below we explain the methods addressing these issues.

Unfolding transformation-specific language

As explained above, applying a transformation yields a goal within a language that might be specific for use of the transformation tool. For this matter we simply use Isabelle's `unfold` tactic, which unfolds any appearance of an unwanted constant into its definition.

Discarding false representations A single structural transformation may induce a variety of possible transformations to a sentence. Some of the induced transformations may lead to false subgoals. Then, we should exclude these false steps from the search, as part of the postprocessing of a transformation.

This is particularly relevant for combinatorial proofs, where it is not enough to take any representative set for every natural number. The most difficult step in this kind of proofs is often choosing the representative sets wisely. Take, for example, the case of Pascal's formula:

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$$

The left-hand side number is the cardinality of $Z = \{s \in \texttt{Pow}\,A : |s| = k+1\}$, where A is a set with cardinality $n+1$. It is standard to make $A = \{0, \ldots, n+1\}$. Naively, we can choose representatives for $\binom{n}{k}$ and $\binom{n}{k+1}$ as $X = \{s \in \texttt{Pow}\,B : |s| = k\}$ and $Y = \{s \in \texttt{Pow}\,B : |s| = k+1\}$ respectively, with $B = \{0, \ldots, n\}$. However we cannot prove that $Z = X \cup Y \wedge X \cap Y \neq \emptyset$, simply because $Z = X \cup Y$ is false; all the elements of Z have cardinality $k+1$, but the elements of X have cardinality k. Moreover, it is easily provably false, which means that either of Isabelle's counterexample checkers, Quickcheck [5] or Nitpick [2], will find the counterexample (in this case, the empty assignment).

The combinatorial proof to Pascal's formula comes from choosing X as $\{s \cup \{n+1\} \in \texttt{Pow}\,B : |s| = k\}$. Thus, if the choice $X = \{s \cup \{n+1\} \in \texttt{Pow}\,B : |s| = k\}$ is the second option in the induced transformations, simply discarding the first (false) option, yields the desired transformation.

The tactic `rerepresent_tac`

Preprocessing, transferring and postprocessing require a transformation to have the following information:

- A set of transfer rules `tr`.
- A set of pre-processing definitions and elimination rules (`pret`, `elim`).
- A set of post-processing definitions `postt`.
- A predicate representing the range where a transformation can be applied: $\lambda \mathtt{x}.\, \mathtt{P}\, \mathtt{x}$.

Thus, when setting up any specific transformations, all of the items of the list must be defined.

The overall design of `rerepresent_tac` consists of the following steps:

1. Apply `split_for_transfer` using $\lambda \mathtt{x}.\, \mathtt{P}\, \mathtt{x}$, handing the untransformed subgoals to `auto`.
2. Apply `tnormalise` using (`pret`, `elim`).
3. Apply the `transfer` mechanism using `tr`.
4. Apply `unfold` using `postt`.
5. Discard the results where Quickcheck or Nitpick find a counterexample.

Furthermore, we have found that simple heuristics at various steps can greatly improve the performance, but we discuss that in sections 6.4 and 7 concerning search and evaluation.

6.4 Search

We present the tactic `representation_search` that searches the space of representations to reach a user-specified end-point, provided that there exists a valid path from source to target. We will briefly mention some simple heuristics that enhance the performance of the tactic.

The tactic `rerepresent_tac` is used to generate new nodes in the search tree. Recall that it induces more than one transformation per sentence. Then, the search of representation involves searching also between potentially many results of a single transformation. Let $\mathscr{T}_1, \mathscr{T}_2, \ldots, \mathscr{T}_n$ be transformations. Suppose that, when applied to a state, each produces

$$\begin{aligned}
\mathscr{T}_1 &\longmapsto s_{(1,1)}, s_{(2,1)}, s_{(3,1)}, \ldots \\
\mathscr{T}_2 &\longmapsto s_{(1,2)}, s_{(2,2)}, s_{(3,2)}, \ldots \\
&\vdots \\
\mathscr{T}_n &\longmapsto s_{(1,n)}, s_{(2,n)}, s_{(3,n)}, \ldots
\end{aligned}$$

In order to have access to the whole set of options (transformations of the goal statement), the results need to be presented as sequence that enumerates them all, such as that given by the lexicographic order:

$$s_{(1,1)}, \ldots, s_{(1,n)}, s_{(2,1)}, \ldots, s_{(2,n)}, s_{(3,1)}, \ldots, s_{(3,n)}, \ldots$$

This is essential, given that each superstructural transformation may induce an infinite number of transformations for a single statement. If we tried to visit all the results of a single transformation before proceeding to the next, it might take an infinite amount of time before getting to the first results of the next transformation.

This establishes the initial breadth-order of the tree (which is not necessarily the same as the order of the *search*; this is discussed later). Figure 2 shows roughly how the first two levels of the search tree look like, assuming there are n transformations.

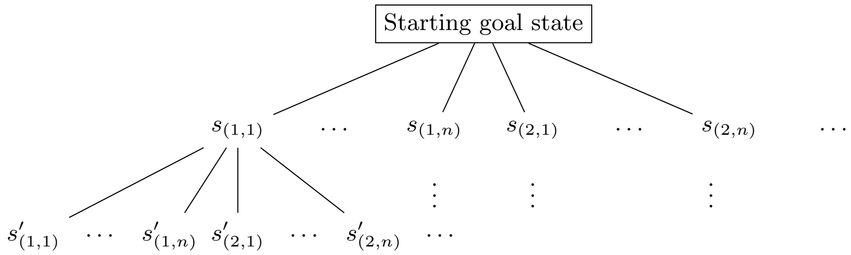

Fig. 2. The figure may wrongly suggest that every transformation generates a non-empty sequence. In fact, most transformations cannot be applied to one particular state, because the statement may concern a superstructure that has nothing to do with the transformation. Of course, our tactic only applies transformations that can be applied.

Search strategy

In figure 2 we represent the tree in which the search is done. This assumes an implicit preference for transformation $\mathscr{T}_i$ over $\mathscr{T}_j$ when $i < j$, and and an even stronger preference for the first results of each transformation over the ones appearing later. Although we have argued that ordering like this is necessary (to avoid get stuck in one transformation), it is not sufficient for an efficient search. Thus, the interleaved sequence is our starting point, but the actual search strategy is slightly more sophisticated.

The tactic `representation_search` works best with best-first search, with a relatively simple heuristic based on size. Depth-first or iterative-deepening work almost as well, but the arbitrary order in which the transformations are applied plays a bigger role in these cases. This is undesirable in terms of reliability. We have found it to be detrimental to the search time and final result in a number of cases. Thus, we implemented a heuristic for the search tactic.

Specifically, we use the following measurements:

1. number of subgoals,
2. the sizes of the subgoal terms,

3. the number of constants appearing in all the subgoal terms,
4. the number of ground types appearing in all the subgoal terms.

For each measure, smaller is always assumed to be better. The actual heuristic is simply the lexicographic order of these 4 measures, preferring them in the order presented here ((1) is preferred over (2), and so on).

A node may have an infinite sequence of children. Thus, the heuristic is only used to order a finite quantity (limited by time consumption). The user may chose the time limit that determines this. Furthermore, the heuristics are applied not only for the search, but also for the presentation of the results (a sequence of results that satisfy the *goal condition*); smaller results are presented first to the user. It should also be noted that, without the heuristics, the transformations are calculated *lazily* (later elements of the sequence are not actually computed until their values are requested). Laziness is unbroken by the interleaving operation, but broken by the calculation of the heuristic; to assess the size of the results we need to compute them. Then, the time limit determines how the balance is set in the trade-off between *lazy* search and *good* (heuristic-driven) search. Lazy/depth-first search will usually be faster but not necessarily (e.g., larger statements take longer to transform), and heuristic search will usually yield better results, and will sometimes be faster (because smaller statements are faster to transform).

We have set the *goal condition* to be based on a set of ground types $\{\tau_1, \ldots, \tau_m\}$ provided by the user. Specifically, the condition is that all types of the set appear in the statement of the result. The user specifies this set when invoking the tactic `representation_search`. This condition simply restricts the results to those with an interpretation in a desirable superstructure. Thus, if the user wants to find whether a statement about numbers can be transformed into a statement about multisets, they may simply provide $\{\alpha$ `multiset`$\}$ when applying `representation_search`.

6.5 Summary

We have presented a set of tools for searching the space of representations. Our main tactic, `rerepresent_tac`, has what we consider the minimum necessary requirements for applying single transformations. The tactic `representation_search` searches the space of representations using `rerepresent_tac` to generate the nodes. The results of using these are presented in section 7.

7 Experiments, Evaluation and Future Work

The main technical contributions presented in this paper are:

- The mechanisation in Isabelle of various transformations observed in discrete mathematics, plus a method for automatically constructing the reverse transformations.
- A number of formal proofs using these transformations.

– A tool for automatically searching the space of representations to reach a user-specified end-point.

To asses their value, they should be seen in the light of our hypothesis:

> *Change of representation, via structural transformations, is a valuable tool for the construction of proofs. In the context of interactive computer mathematics, the value stems mainly from the reduction of effort required from the user, and the quality of the proofs produced.*

Thus, it concerns the level of human interaction, and the quality of the proofs produced. First, we need to discuss how the quality of proofs can be assessed.

How do we rate proofs?

The following measures are good candidates for rating a proof:

- Length: shorter proofs are generally better than longer proofs, though not necessarily.
- Conceptual ease/readability: more understandable proofs are generally better, but this depends on *who* tries to understand the proof.
- Uniformity/generality: if the methods used are more general, the proof may have value for reuse/learning.

The three points are not independent, nor are they necessarily positively related. For example, short proofs may be hard to understand if the reduction in length is due to obviating crucial or complicated steps. However, added length due to unnecessary steps also lowers readability. Conceptual ease and uniformity are also positively related, but not necessarily. For example, proofs formed by obscure decision procedures are as uniform as they can be, but may not be readable.

Textbook proofs are one kind of prototypical high-quality proof, compromising between the three points, sometimes focusing a little more on conceptual ease/readability, and sometimes on uniformity/generality, depending on the didactic purpose (convincing the reader, teaching basic concepts, teaching some proof technique, etc.).

In light of this, we can discuss a bit further how tactics can be assessed.

How do we rate reasoning tactics?

The following measures are good candidates for rating a reasoning tactic:

- Rating induced by the proofs in which the tactic is used: with the rating of proofs mentioned above.
- The amount of effort required from the user: better tactics reduce the amount of effort.
- Range of applications: solving more problems is obviously desirable.
- Consumption of resources: using less memory and time is better. We could extend this to include 'theory' as a resource, e.g., using more lemmas or requiring a stronger theory consumes a certain kind of resource.

As stated in our hypothesis, the standard for new methods of reasoning is that they reduce the amount of human interaction for some collection of proofs. In the best case scenario, the level of interaction is reduced to none. In terms of the above rating points, this is accounted for by the effort measure and the range of applications.

A comparison issue

When trying to rate tactics and proofs an issue arises. There are no universal standards of comparison. For example, suppose one wants to assess whether a tactic is valuable. Unless the tactic automatically solves the problem, one has to look at proofs using this tactic. However, what should they be compared to? If the evaluator cannot find a proof without the tactic it does not mean there is none. If the evaluator can find only a longer proof without the tactic it does not mean there is no shorter one (should the problem be given to many mathematicians to get a statistical result?). Unless there is a proof showing that the tactic is necessary, the value is questionable in principle.

With all this in mind, let us analyse the results of applying our tactics to some problems.

7.1 Case analysis (prime factorisation method)

Recall the problem presented in table 1 from section 3 (here with the Isabelle syntax):

$$\exists\, x\, y\, z : \mathtt{nat}.\ \mathtt{gcd}\, x\, y \neq 1 \,\wedge\, \mathtt{gcd}\, x\, z \neq 1 \,\wedge\, \mathtt{gcd}\, y\, z \neq 1 \,\wedge\, \mathtt{gcd}\, x\, (\mathtt{gcd}\, y\, z) = 1$$

For this example we will present the results of applying no transformation, applying the tactic `rerepresent_tac` (which takes only one transformation and applies it), and applying the tactic `representation_search` which takes a set of transformations and searches the space using them.

The first thing to know is that, in Isabelle-2015, none of the automatic methods can solve it directly (including all the external provers called by Sledgehammer). This is very surprising, given that assigning random values to x, y and z should eventually produce a solution (which in principle is much easier for a computer than for humans). Then the user has to provide the values (6,10 and 15 work, for instance). Once the user provides them, the proof can be finished by tactic `eval`, which happens to know an algorithm to calculate the greatest common divisor. Then, the proof without a transformation is trivial, but the user still has to provide the values.

So how about applying a transformation? Simply applying the tactic `rerepresent_tac` with BN_transformation (numbers-as-bags-of-primes) we get the following subgoals:

- `bag_of_primes` $?x$
- `bag_of_primes` $?y$
- `bag_of_primes` $?z$

- $?x \cap ?y \neq \{\}$
- $?x \cap ?z \neq \{\}$
- $?y \cap ?z \neq \{\}$
- $?x \cap (?y \cap ?z) = \{\}$

where variables preceded by '?' are variables which need to be instantiated (also, they are of type multiset). Again, the automatic methods cannot find the right instantiations, but once the user provides them, the system can find proofs ($\{2, 3\}$, $\{2, 5\}$ and $\{3, 5\}$ work). However, this time it is not as simple as applying tactic `eval`, but the external provers find the proof using various basic lemmas about multisets[13]. Moreover, there are more (trivial) subgoals that need to be proved (the first three).

Thus, the proof after the transformation is more lengthy and consumes more resources in every respect. However, it is crucial to notice that the construction of the multisets is motivated; guided (add one element to each pair and not to the remaining element, see figure 3), whereas the only motivation or guidance we can think for constructing the numerical examples (6, 10 and 15) is by thinking about them in terms of their prime factors i.e., applying the transformation![14] In terms of our rating points, it can be argued that conceptual ease is higher using the transformation.

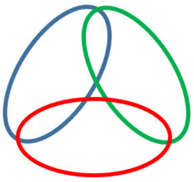

Fig. 3. Add a different element to each of the pairwise intersections.

For this example the evaluation is for the tactic `rerepresent_tac` which involves only one transformation and does not search the space. However, things get better if we use the tactic `representation_search`, with a catalogue that includes transformations `BN_transformation` and `NP_transformation` (paramet-

[13] We believe this is due to the theory of multisets being under-developed in Isabelle, compared to number theory, even though multisets can be argued to be intuitive, common-sense objects in human reasoning. Interestingly, proving the subgoal $\{2, 3\} \cap \{2, 5\} \neq \{\}$ is harder than the others (requires more lemmas and takes much more time to the external provers), even though it is clearly analogous. The difference is that the intersection is the first element of both multisets. This is further evidence that the theory of multisets in Isabelle needs more development to resemble the human intuitions we have about multisets.

[14] Thus, it should be concluded that there is an interesting motivation, and research regarding the use of transformations for the construction of examples and counterexamples.

ric multiset transformation, specifically regarding a bijection between naturals and primes).

Recall that the end-point of the search is determined by the user. If we select the multiset structure as the desired end-point we get:

- $?x \cap ?y \neq \{\}$
- $?x \cap ?z \neq \{\}$
- $?y \cap ?z \neq \{\}$
- $?x \cap (?y \cap ?z) = \{\}$

were the variables are of type `multiset`. The reason why the subgoals `bag_of_primes` $?x$ disappear is that the transformation `NP_transformation` bijects primes with natural numbers. Furthermore, this transformation preserves the multiset operations because it is bijective (we only need to find isomorphic solutions for figure 3). Moreover, the heuristic of preferring states where the size of statements is smaller makes sure we end with this.

This proof is slightly better than the previous in length, but not much in resource consumption; is still requires reasoning about multisets, which we have argued is not ideal.

But let us stop for a minute to think more about why our intuitions might seem so simple in spite of being in a clunky theory about multisets. Is it not because we are actually reasoning about sets, and we know that it is equivalent? Then, what if we include the transformation `SMi_transformation` (set-in-multiset transformation), which injects all finite sets into the multiset type. Indeed, when we search the space of transformations using `BN_transformation`, `NP_transformation` and `SMi_transformation`, our search tactic can find more results.

If we simply select the structure of sets as an end-point, with the above 3 transformations as options, we get the following:

- `finite` $?x$
- `finite` $?y$
- `finite` $?z$
- $?x \cap ?y \neq \{\}$
- $?x \cap ?z \neq \{\}$
- $?y \cap ?z \neq \{\}$
- $?x \cap (?y \cap ?z) = \{\}$

where the variables are of type `set`. In this case, providing the values ($\{0,1\}$,$\{1,2\}$ and $\{0,2\}$ work[15]) makes everything solvable by tactic `simp` (Isabelle's ubiquitous simplification tacic). In this case the consumption of resources is much lower and arguably it is much better in terms of conceptual ease. In fact, one might argue that this representation, in terms of sets, was the best interpretation of figure 3.

[15] It should be noted that, in general, this transformation eliminates potential solutions because not all multisets map to sets. In other words, finding a set solution guarantees that there is a multiset solution, but not the other way around.

Moreover, and very interesting to note, is that even though the user still has to provide the values for the variables, we got something different when trying the following: negate the result, and then run a counterexample checker. Nitpick finds the right instantiations. In fact, it finds precisely $\{0, 1\}$, $\{1, 2\}$ and $\{0, 2\}$. This means that the mechanisms for constructing counterexamples for sets with these specific constraints are there (in Isabelle) but they are not implemented in any of the automatic provers to find examples! Thus, not only is the consumption of resources and conceptual ease better, but if the mechanisms the counterexample checkers are using for finding counterexamples were implemented for normal existential proofs in the typical provers, we would be in the scenario where our tactic almost solves the problem.

Other applications in number theory

We have also constructed the mechanical proof of the other problem in table 1 (using `representation_search`), with similar results. In total, we have mechanised 5 proofs of number theory problems using `representation_search`, where `BN_transformation` is used, followed by other transformations.

In particular, we mechanised a proof of problem 1 from section 3. This example is interesting; we believe it is conceptually easy, and the end result is solved by a decidable fraction of arithmetic. However, the steps between the application of the transformation and reaching the decidable expression are surprisingly tedious. The tactic `representation_search` finds a two-step transformation, starting with `BN_transformation`, followed by `NP_transformation`. The resulting proof after that consists of 14 tactic applications, including two uses of our tactic `rerepresent_tac` with transformation `FM_transformation`. This is how the problem eventually gets reduced to one about natural numbers (the multiplicities of the multisets, represented as the values of $\mathbb{N}$-valued functions).

The extensive use of transformations in the proof of problem 1 seems to highlight the advantages of our reasoning methods (the mechanisation of the transformations, plus the tactics that facilitate their use). However, it remains difficult to evaluate whether a simpler proof can be constructed without them. We have none, but that does not mean none exists.

7.2 Case analysis (combinatorial method)

The following analysis involves combinatorial proofs (double counting). The analysis in this case is much more complicated, because the proofs in question require plenty of human interaction. However, there is value in the fact that our tactic (plus the transformation) works as an implementation of the combinatorial method of proof. The following has been said about these proofs and results: *'one of the most important tools in combinatorics'* by [21, p.4], and *'The proofs of these identities are probably even more significant than the identities themselves'* by [4, p.65]. Their ubiquity and power puts these proofs in the category of 'valuable by uniformity' and 'valuable by conceptual ease' (although these examples suggest conceptual ease and readability are different things, as we will see).

These proofs do not involve search of the space of representations, but rather just search within one transformation.

Recall Pascal's identity:

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$$

A combinatorial proof involves dividing the set of $(k+1)$-subsets of $\{0,\ldots,n\}$ into two parts: those which contain n and those that do not.

As mentioned in section 6.3, the correct transformation of this statement requires discarding transformations which lead to a false subgoal, e.g., the right set representatives of numbers have to be found. Our tactic `rerepresent_tac` does it in this case and that is enough for the right representation to be found.

The resulting combinatorial proof has a lot of human interaction (a structured proof with 12 applications of tactics with no identifiable pattern, requiring the generation and proof of one general lemma). The comparison with a proof without a transformation is embarrassing at first sight; without a transformation the theorem is solved by `simp`. However, a simple inspection shows why this is: in the Isabelle library, the `choose` operator is defined recursively by Pascal's identity. However, there are many equivalent ways of defining it (often as $\frac{n!}{k!(n-k)!}$ or as the cardinality of a specific set). This points at another important problem for evaluating tactics and proofs: it all depends on the formalisms chosen and the background knowledge.

The next example has a more positive analysis. Also from section 3, recall the identity:

$$\sum_{0\leq i\leq n} \binom{n}{i} = 2^n$$

In [4, p.66] the combinatorial proof is given in less than 3 lines, in a perfectly elegant and intuitive way. In Isabelle, after applying `rerepresent_tac` we can obtain a proof in 7 sequential tactic applications using 10 basic lemmas regarding finite sets from the Isabelle library (they do not have to be provided by the proof; the external provers provide them). This is almost an automatic proof, with a simple pattern: first some definitions are unfolded, then tactic `safe` is applied (a safer, more restricted variation of `auto`). After this, the external theorem provers suggest the application of the tactics to solve each of the 5 remaining subgoals. Thus, the only interaction by the user is calling the external provers. So, in fact, the interaction of the user consists of two steps, which are very usual as 'first steps' in interactive mechanical proofs: unfolding strange (unfamiliar or unusual) definitions that may appear, followed by applying `auto` or `safe`.

Interestingly, there is a proof for this theorem in the Isabelle library, which provides us with something to compare ours. This is another well known (and quite beautiful) proof. It is done by instantiating the binomial theorem

$$(a+b)^n = \sum_{0\leq i\leq n} \binom{n}{i} a^i b^{n-i}$$

with $a = b = 1$. After the user-provided instantiation the proof is immediate but it requires the binomial theorem, whose formal proof is very interactive and complex (see the Binomial theory in Isabelle 2015). Still, the binomial theorem is important, but if we are concerned with how theories develop from the bottom up we cannot take it for granted. Without the binomial theorem the proof that remains is by induction on n. To our knowledge no one has yet implemented this proof in Isabelle, at the point of writing.

The examples in which we have applied this transformation (all the problems in table 2 of section 3) are simple but essential and representative of the overall technique of combinatorial proof. The general proof technique has a really extensive range of applications. We believe that our tactic, as a foundation, can be used for all sorts of problems in this range. For this, the search and selection of representative sets will probably need to be improved (recall that our selection relies on counterexample checking and preference of smaller subgoal statements), as well as the techniques for reasoning about finite sets.

Final remarks on evaluation

We gave an analysis of the use of the tactics `rerepresent_tac` and `representation_search`, in a couple of families of proofs.

For the first family of proofs, we showed the potential of searching the space of representations for constructing examples (or counterexamples). This opens up the potential for techniques like this one to be used widely (increasing its value by broadening the range of its applications). In general, the resulting proofs are human-readable and intuitive, but not necessarily cheaper computationally.

For the second family of proofs, we showed how our tactic is an implementation of a very general proof technique in combinatorics. Particularly, the application of counterexample checking to discard false representations is a valuable tool for choosing the right set representatives.

In general, the time consumption of the tactic applications is between a few milliseconds and a few seconds (for the search tactic, when there is a large search space). We did not focus specifically on optimising time. This is a challenge for future work.

Our experiments and evaluation are only restricted to a small part of discrete mathematics, although we have formalised transformations (with no examples of their use, yet) for a slightly larger (but not exhaustive) part. Simply in this area of mathematics we have identified plenty of potential applications of these techniques, and we foresee and encourage its use in different areas.

8 Related Work

Although representation is widely recognised as a crucial aspect of reasoning, to our knowledge there has been no attempt to incorporate the *automatic* search of representation into reasoning tools.

Theory interpretations. 'Little Theories' is the notion that reasoning is best done when it is modular [6]. IMPS is an interactive proof system implemented based on the principles of Little Theories [7]. The modules, or *little theories* of IMPS are small axiomatic theories connected by *theory interpretation* to allow reuse of theorems (passed down through interpretation).

The little theories approach is captured in Isabelle by the use of *locales* [1], and their applications for type classes [10].

The typical uses of theory interpretations for problem solving can be captured easily with the traditional reasoning techniques (as the uses of locales in Isabelle demonstrate). However, theory interpretations do not capture (at least immediately) the transference of theorems across object-level transformations (conceptualised here as superstructural transformations). The connections between theory interpretation and theorem transference across object-level transformations is an interesting avenue of research, outside of the scope of this paper.

Logic translations. The concept of *institution* was introduced as a general notion of logical system [8]. The Heterogeneous Tool Set (HETS) [17], based on the theory of institutions, was developed mainly to manage and integrate heterogeneous specifications. It links various logical systems, including some of Isabelle's different logics, and provides a way of translating between them. We do not know of any applications of HETS specifically taking advantage of heterogeneity as a means of finding proofs under one representation where other representations fail.

Some notable applications of logic translations are from higher-order logics (commonly used for *interactive* theorem provers) into first-order logics (commonly used for *automated* theorem provers). Many powerful tools in Isabelle, such as Sledgehammer [3] and Metis [16] make use of these translations. Others [13] have called for translations of this kind (and other kinds of *problem reformulations*) to be implemented as tactics (for interactive use) in theorem provers. It is unclear how *inter*-logical translations could be built as tactics in systems like Isabelle, but we have seen how some kinds of *intra*-logical reformulations can be implemented as tactics with tools like the Transfer package's.

Other uses of the Transfer package. The use of the Transfer package has changed how new quotient types and 'subtypes' are defined. This is what the *Lifting* package does [11]. As part of the lifting package, there is a way of automatically transferring definitions from an old (raw) type to a new (abstract) type (e.g., multisets are defined as an abstract type from the type of $\mathbb{N}$-valued functions).

The lifting of definitions and transference of theorems from raw to abstract types has been the main application of the Transfer package, although the generality of their approach is acknowledged by the developers. Embodying this generality, they have demonstrated the transference from integers to natural numbers through the 'inclusion' relation. Another application of the transfer package is refinement used by Isabelle's code generator [9].

Our work adds a few transformations which correspond to important reasoning techniques in basic discrete mathematics, and extends the transfer tools for the automation of the search of representation.

9 Conclusion

We have given motivation and evidence for the hypothesis *that change of representation is valuable for the construction of proofs*. Specifically, we presented a mathematical framework and a set of tools which extend Isabelle's Transfer package. We have used these tools to test the hypothesis and found moderate, but promising results. We have shown a potential for the application of tools which automate the search of representation in various aspects of reasoning. In particular, we showed some potential in the construction of examples (and counterexamples) in number theory, as well as a potential for applications in proofs using the combinatorial proof technique. These are interesting ventures for future work.

Bibliography

[1] Clemens Ballarin. Interpretation of locales in Isabelle: Theories and proof contexts. In *Mathematical Knowledge Management*, pages 31–43. Springer, 2006.

[2] Jasmin Christian Blanchette and Tobias Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. *ITP*, 6172:131–146, 2010.

[3] Sascha Böhme and Tobias Nipkow. Sledgehammer: judgement day. In *Automated Reasoning*, pages 107–121. Springer, 2010.

[4] Miklós Bóna. *A walk through combinatorics: an introduction to enumeration and graph theory.* World scientific, 2011.

[5] Lukas Bulwahn. The new Quickcheck for Isabelle. In *Certified Programs and Proofs*, pages 92–108. Springer, 2012.

[6] William M Farmer, Joshua D Guttman, and F Javier Thayer. Little Theories. In *11th International Conference on Automated Deduction*, pages 567–581. Springer, 1992.

[7] William M Farmer, Joshua D Guttman, and F Javier Thayer. IMPS: an interactive mathematical proof system. *Journal of Automated Reasoning*, 9(11):213–248, 1993.

[8] Joseph A Goguen and Rod M Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM (JACM)*, 39(1):95–146, 1992.

[9] Florian Haftmann, Alexander Krauss, Ondřej Kunčar, and Tobias Nipkow. Data refinement in Isabelle/HOL. In *Interactive Theorem Proving*, pages 100–115. Springer, 2013.

[10] Florian Haftmann and Makarius Wenzel. Constructive type classes in Isabelle. In *Types for Proofs and Programs*, pages 160–174. Springer, 2006.

[11] Brian Huffman and Ondřej Kunčar. Lifting and Transfer: a modular design for quotients in Isabelle/HOL. In *Certified Programs and Proofs*, pages 131–146. Springer, 2013.

[12] Joe Hurd. System description: The Metis proof tactic. *ESHOC*, pages 103–104, 2005.

[13] Manfred Kerber and Axel Präcklein. Using tactics to reformulate formulae for resolution theorem proving. *Annals of Mathematics and Artificial Intelligence*, 18(2-4):221–241, 1996.

[14] Ramana Kumar, Rob Arthan, Magnus O Myreen, and Scott Owens. HOL with definitions: Semantics, soundness, and a verified implementation. In *Interactive Theorem Proving*, pages 308–324. Springer, 2014.

[15] Ondřej Kunčar and Andrei Popescu. A consistent foundation for Isabelle/HOL. In *Interactive Theorem Proving*, pages 234–252. Springer, 2015.

[16] Jia Meng and Lawrence C Paulson. Translating higher-order clauses to first-order clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008.

[17] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, HETS. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 519–522, 2007.

[18] Tobias Nipkow, Lawrence C Paulson, and Makarius Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic.* Springer-Verlag, 2013.

[19] Lawrence C Paulson and Jasmin Christian Blanchette. Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers. *PAAR@IJACR*, 2010.

[20] Matthieu Sozeau. A new look at generalized rewriting in type theory. *J. Formalized Reasoning*, 2(1):41–62, 2009.

[21] Jacobus Hendricus van Lint and Richard Michael Wilson. *A course in combinatorics.* Cambridge university press, 2001.

[22] Tjark Weber. SMT solvers: New oracles for the HOL theorem prover. *International Journal on Software Tools for Technology Transfer 13.5*, pages 419–429, 2011.

[23] Théo Zimmermann and Hugo Herbelin. Automatic and transparent transfer of theorems along isomorphisms in the Coq proof assistant. *arXiv preprint arXiv:1505.05028*, 2015.