Abstract

In their article titled *From Types to Sets by Local Type Definitions in Higher-Order Logic* and published in the proceedings of the conference *Interactive Theorem Proving* in 2016, Ondrej Kunčar and Andrei Popescu propose an extension of the logic Isabelle/HOL and an associated algorithm for the relativization of the *type-based theorems* to more flexible *set-based theorems*, collectively referred to as *Types-To- Sets*. One of the aims of their work was to open an opportunity for the development of a software tool for applied relativization in the implementation of the logic Isabelle/HOL of the proof assistant Isabelle. In this article, we provide a description of a software framework for the interactive automated relativization of definitions and theorems in Isabelle/HOL, developed as an extension of the proof language Isabelle/Isar, building upon some of the ideas for further work expressed in the original article on Types-To-Sets by Ondrej Kunčar and Andrei Popescu and the subsequent article *Smooth Manifolds and Types to Sets for Linear Algebra in Isabelle/HOL*, which was written by Fabian Immler and Bohua Zhan and published in the proceedings of the *International Conference on Certified Programs and Proofs* in 2019.

**CCS Concepts:** • Theory of computation → Higher order logic.

**Keywords:** Proof Assistants, Isabelle, Higher Order Logic, Formalization of Mathematics

1 Background

Isabelle [33] is one of the leading interactive proof assistants; it is generic and supports several formal calculi [35], with Isabelle/HOL being arguably the most important one. Isabelle/HOL is a classical Higher-Order Logic with rank-1 polymorphism, Hilbert Choice, the Axiom of Infinity, axiomatic type classes, ad hoc overloaded constant definitions and type-definitions (e.g., see [23]).

The *standard library* [13] that is associated with the object logic Isabelle/HOL and provided as a part of the standard distribution of Isabelle contains a significant number of formalized results from a variety of fields of mathematics. Nevertheless, using the argot that was promoted in the original publication of Types-To-Sets [22], the formalization is performed using a type-based approach. Thus, for example, the carrier sets associated with algebraic structures consist of all terms of an arbitrary type. This restriction can create an inconvenience when working with mathematical objects induced on a subset of the carrier set associated with the original object (e.g., see [16]). To address this limitation, several additional libraries were developed upon the foundations of the standard library (e.g., [4] and [12]). In terms of the argot associated with Types-To-Sets, these libraries provide the set-based counterparts of many type-based theorems in the standard library. Nonetheless, the proofs of the theorems that are available in the standard library are restated explicitly in the libraries that contain the set-based results. This unnecessary duplication of efforts is one of the primary problems that the framework Types-To-Sets is meant to address.

The framework Types-To-Sets offers a unified approach to structuring mathematical knowledge formalized in Isabelle/HOL: it allows for every type-based theorem to be relativized (converted to a set-based theorem) using a pre-defined relativization algorithm, while articulating the relationship that exists between such type-based theorems and their set-based counterparts clearly and explicitly [22]. However, [22] is a work of logic — it offers next to no tools for harnessing the power of the framework that it describes in any given proof assistant [16]. In this article, we describe a particular implementation of the framework Types-To-Sets for Isabelle/HOL in Isabelle/ML [30, 43] that takes the form of an extension of the language Isabelle/Isar [41, 42] with several new commands [44]. Furthermore, we provide several application examples, demonstrating the capabilities of the proposed software and the framework itself.
2 Contributions and Structure

In this article, we describe two closely related prototype software frameworks for Isabelle/HOL: Conditional Transfer Rule (CTR)\(^1\) and Extension of Types-To-Sets (ETTS). The primary objective for the development of the CTR and the ETTS was to take Types-To-Sets from a successful thought experiment in type theory to one of the standard tools for library design through automation and integrated with Isabelle/Isar and Isabelle IDE. We list the primary contributions of this article and the software that it describes below:

- The software framework CTR that provides implementations of several algorithms for the automation of unoverloading and relativization of constants (see Subsection 3.3 and Section 5), presented in the form of Isabelle/ML modules, and complemented with an extension of the language Isabelle/Isar with the commands ud and ctr that serve as an interface for the invocation of the aforedescribed functionality.
- Several amendments to the relativization algorithm proposed in [22] that introduce convenience features for the relativization of theorems in a local context and tools for post-processing of the result of the relativization, drawing inspiration from the study in [16].
- The software framework ETTS that provides an implementation of the augmented relativization algorithm, presented in the form of Isabelle/ML modules, and complemented with an extension of the language Isabelle/Isar with the commands tts_context, tts_lemmas and tts_lemma that provide an interface for the application of the relativization algorithm.
- An extensive collection of the application examples of the ETTS that includes a remake of the relativization study performed in [16] (relativization of over 200 theorems in the area of linear algebra) and a novel large-scale application of the ETTS to elements of the standard library (relativization of over 800 theorems in the areas of abstract algebra and general topology).

The CTR and the ETTS are available from the Archive of Formal Proofs under the terms of the 3-clause BSD license [28, 29]. The remainder of this article is organized as follows:

- In Section 3, we provide a motivating example.
- In Section 4, we present a review of the previous work.
- In Section 5, we present a description of the CTR.
- In Section 6, we present a description of the ETTS.
- In Section 7, we present several application examples of the ETTS.
- In Section 8, we provide a summary of our work and suggestions for further work.

Before we proceed, we note that we tried to make this article self-contained, but the reader could benefit from being familiar with [22] or [24].

3 A Motivating Example

3.1 Background

We begin our discussion by providing an example that demonstrates some of the capabilities of the framework Types-To-Sets but also presents some of the weaknesses of its current implementation. Moreover, we show how the software framework that is proposed in this article addresses and largely eliminates the aforementioned weaknesses.

3.2 Elementary Point-Set Topology in Isabelle/HOL

In the conventional setting of informal set theory, a topological space is a pair \((U, \tau)\) that consists of the underlying set \(U\) and a collection of sets \(\mathcal{T} \subseteq \mathcal{P}(U)\) (where \(\mathcal{P}\) denotes the operation of taking a power set) subject to certain further conditions.\(^2\) The collection \(\mathcal{T}\) can also be identified with a unary predicate \(\tau\) on the subsets of \(U\) such that \(\tau(A)\) and only if \(A \subseteq U\). In what follows, we use \((U, \tau)\) as the definition of a topological space, as we believe that this form is more compatible with the type-theoretic setting of Isabelle/HOL (at least, a similar form of the definition is used in the standard library). In this case, the aforementioned defining conditions can be expressed as follows:

- \(\tau(U)\)
- \(\forall a, b \subseteq U . \tau(a) \rightarrow \tau(b) \rightarrow \tau(a \cap b)\)
- \(\forall A \subseteq \mathcal{P}(U) . (\forall a \in A. \tau(a)) \rightarrow \tau(U)\)

Given a topological space \((U, \tau)\), the sets \(a \subseteq U\) such that \(\tau(a)\) are called open sets and \(\tau\) is referred to as a topology on \(U\). Given a topological space \((U, \tau)\), a set \(A \subseteq U\) is closed if and only if its relative complement \(U - A\) is open. A pair \((V, \tau_V)\) is a subspace of a topological space \((U, \tau)\) if and only if \(V \subseteq U\) and \(\tau_V\) is a unary predicate such that \(\tau_V(A)\) if and only if \(a = V \cap b\) for some \(b \subseteq U\) such that \(\tau(b)\). Of course, a subspace of a topological space is also a topological space [19].

There exist many methodologies for the definition of a topological space in the type-theoretic setting of Isabelle/HOL. Most such definitions rely on the built-in unary type constructor set such that any (well-typed) term \(\alpha\) set represents a collection of terms of the type \(\alpha\) and the built-in nullary type constructor bool (which will be denoted as \(\mathbb{B}\) for conciseness) that represents truth values [21, 23, 31].\(^3\) For example, we may define a constant [21, 23] \(\tau_{\alpha\sigma}\) set : \((\alpha \text{ set } \rightarrow \mathbb{B})\) \rightarrow \mathbb{B}\) such that for any variable \(\tau_{\alpha\sigma}\) set, \(\tau_{\alpha\sigma}\) set \(\tau\) holds if and only if the following axioms are satisfied:

- \(\tau\) (UNIV_\(\alpha\) set)
- \(\forall a_{\alpha\sigma}\text{ set} . \exists b_{\alpha\sigma}\text{ set} . \tau a \rightarrow \tau b \rightarrow \tau (a \cap b)\)
- \(\forall A_{\alpha\sigma}\text{ set} \subseteq \mathcal{P}(U) . (\forall a \in A. \tau a) \rightarrow \tau (U)\)

\(^1\)Appendix C provides a list of acronyms.

\(^2\)We omit a detailed review of the subject and refer our readers to standard textbooks, such as [6, 19], for further information.

\(^3\)The type \(\sigma\) of a term can be indicated using either the colon notation \((t : \sigma)\), the subscript notation \((t_{\sigma})\) or even omitted entirely if it is deemed that it can be inferred from the context of the discussion [21, 23].
which can be used to identify open sets among the collections of terms of type $\alpha$. Effectively, given any type $\sigma$ and any term $\tau \rightarrow \beta$, the constant $\text{cl}_{\text{with}}$ can be used to indicate that the pair $(\sigma, \tau)$ forms a topological space. Given a topological space $(\alpha, \tau \rightarrow \beta)$ in the aforementioned sense, the constant

$$\text{cl}_{\text{with}} : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta,$$

defined in a manner such that $\text{cl}_{\text{with}} \tau S = \tau (\neg S)$ holds for any $\tau \rightarrow \beta$ and $S : \alpha$ set (in the type-theoretic setting of Isabelle/HOL the absolute complement $\neg S : \alpha$ set of a set $S : \alpha$ set is well-defined and given by $\neg S = \text{UNIV} - S$), can be used to identify closed sets among the collections of terms of $\alpha$. Many theorems of the conventional point-set topology can be proved in this setting. For example, the following theorem expresses a well-known property of closed sets [19]:

$$\text{ts}_{\text{with}} \tau \rightarrow \forall A : \alpha \rightarrow \beta, \text{cl}_{\text{with}} \tau A \rightarrow \text{cl}_{\text{with}} \tau (A \cup B). \tag{1}$$

Nonetheless, it is possible to restate the results above in a more concise format that offers a multitude of other advantages from the perspective of the compatibility with standard proof tools available in Isabelle. This is made possible by two prominent and closely related features of Isabelle/HOL: ad hoc overloaded constant definitions (or simply overloading) and axiomatic type classes (or simply classes). Overloading is a mechanism by which a given constant can be declared with a given type and overloaded on its type instances (i.e., less general types) [23, 40, 44]. Thus, for example, the constant $(*)_{\alpha \rightarrow \beta : \alpha \rightarrow \beta}$ can have the distinct specifications $(*)_{\text{nat} : \text{nat} \rightarrow \text{nat}} = t_1$ and $(*)_{\text{int} : \text{int} \rightarrow \text{int}} = t_2$ for some suitable terms $t_1$ and $t_2$ and types nat and int. The overloading mechanism is subject to several restrictions [23], but we omit further details, as they are less important from the perspective of this article. The classes provide a mechanism by which types can carry implicit assumptions. The classes become most useful when combined with overloading. For example, given the constant declaration $(*)_{\alpha : \alpha \rightarrow \alpha}$, the classes can be used to state that $(*)$ is associative on $\alpha$, which, in turn, can be used to reason about arbitrary semigroups [22]. We use the notation $\alpha : Y$ or $\alpha Y$ to indicate that the given type variable $\alpha$ carries the class constraint $Y$ (more precisely, a non-default single-class sort constraint, i.e., a symbolic intersection of a finite number of classes [9]). Given that a class can be identified with a predicate on types, $\alpha$ may also be used to represent the predicate associated with the class and used like so: $Y (\alpha)$.

Continuing with the example, the standard library provides the class topological_space (abbreviated as ts) that can be used to identify the types that obey the laws of a topological space with respect to the constant predicate $\text{open}_{\text{ts}} : \alpha \rightarrow \beta$, which can be used to identify open sets among the collections of terms of such a type [10]. The constant $\text{closed}_{\text{ts}} : \alpha \rightarrow \beta$ (abbreviated as cl) can be used to identify closed sets among the collections of terms of $\alpha$. The constant is defined as $\text{cl} S = \neg (\neg S)$ for any $S : \alpha$ set. In this setting, the theorem given by Eq. 1 can be stated as

$$\forall A : \alpha \rightarrow \beta, \text{cl}_{\text{with}} \tau A \rightarrow \text{cl}_{\text{with}} \tau (A \cup B). \tag{2}$$

The constants such as $\text{ts}_{\text{with}}$, $\text{cl}_{\text{with}}$ and $\text{cl}$, their definitions and theorems about them (e.g., Eq. 1 and Eq. 2) will be referred to as the type-based constants, the type-based definitions and the type-based theorems, respectively.

Unfortunately, there is no natural translation of the set-theoretic definition of a subspace of a topological space to a type-based set-theoretic definition: the conventional (semantic) subtype relation [44] in Isabelle/HOL does not possess many of the familiar properties of the subset operation from set theory. For example, the same term cannot “belong” to two distinct types due to syntactic restrictions, even if one of these types is a subtype of the other (e.g., see [23]). Perhaps, the closest definition from the conventional point-set topology that can be naturally translated is that of an injective continuous function.

Nonetheless, there exists a solution to the problem outlined in the previous paragraph. As mentioned previously, the type constructor set can be used to single out an arbitrary collection of terms of a given type. Now, the constant $\text{ts}_{\text{with}}^\alpha : \alpha \rightarrow (\alpha \rightarrow \beta) \rightarrow \beta$ could be defined in such a manner that for all terms $U : \alpha \rightarrow \beta$ and $\tau \rightarrow \beta$, $\text{ts}_{\text{with}}^\alpha U \tau$ holds if and only if the following axioms are satisfied:

- $\tau U$
- $\forall A : \alpha \rightarrow \beta, \text{ts}_{\text{with}}^\alpha \tau A \subseteq U, \tau a \rightarrow \tau b \rightarrow \tau (a \cap b)$
- $\forall A : \alpha \rightarrow \beta, \text{ts}_{\text{with}}^\alpha \tau A \subseteq P U, (\forall a \in A, \tau a) \rightarrow \tau (U a)$

($P$ is an abbreviation for the constant $\text{ Pow}_{\alpha : \alpha \rightarrow \beta}$ set that represents the operation of taking a power set in Isabelle/HOL). A subspace of a topological space $(U : \alpha \rightarrow \beta, \tau \rightarrow \beta)$ can be defined as any pair $(V : \alpha \rightarrow \beta, \tau V \rightarrow \beta)$ such that $V \subseteq U$ and $\tau V \rightarrow \beta$ holds if and only if $a = V \cap b$ for some $b \subseteq U$ such that $\tau b$. However, given a topological space $(U : \alpha \rightarrow \beta, \tau V \rightarrow \beta)$, the constant predicate for the identification of closed sets among the subsets of the underlying set has to be redefined as, for example,

$$\text{cl}_{\text{with}}^\alpha : \alpha \rightarrow (\alpha \rightarrow \beta) \rightarrow \beta$$

such that $\text{cl}_{\text{with}}^\alpha U \tau S$ holds if and only if $\tau (S \cap U)$ for all $S : \alpha$ set. Thence, in this setting, the theorem given by Eq. 1 would now look like

$$\text{ts}_{\text{with}}^\alpha U \tau \rightarrow \forall A : \alpha \rightarrow \beta, \text{cl}_{\text{with}}^\alpha \tau A \rightarrow \text{cl}_{\text{with}}^\alpha \tau (A \cup B). \tag{3}$$

The constants such as $\text{ts}_{\text{with}}^\alpha$ and $\text{cl}_{\text{with}}^\alpha$, their definitions and theorems about them (e.g., Eq. 3) will be referred to as the set-based constants, the set-based definitions and the set-based theorems, respectively.
3.3 Types-To-Sets: Purpose and Scope

Despite the problems associated with the type-based reasoning that were exposed in Subsection 3.2, the type-based constants are routinely used in the standard library, as the conventional proof procedures work better for results about such constants and such results can be expressed more concisely (e.g., compare Eq. 2 and Eq. 3) [22].

The set-based definitions and theorems are also used extensively, and entire libraries of set-based results were constructed upon the foundations of the standard library (e.g., see [4] and [12]). Nonetheless, the type-based results cannot be reused in the proofs of the associated set-based results [22]. Thus, despite their semantic similarity, the theorem given by Eq. 3 cannot be inferred directly from the theorems given by Eq. 1 or Eq. 2 in Isabelle/HOL. Arguably, the lack of automation for the translation of the type-based theorems in the standard library to the set-based theorems resulted in the duplication of proof efforts on a dramatic scale. The framework Types-To-Sets was designed precisely to avoid such duplication of efforts by enabling automated translation of the type-based theorems to their set-based counterparts [22]. Thus, the framework enables conversion of the theorems given by Eq. 1 or Eq. 2 to the theorem given by Eq. 3.

In this context, we also provide an assessment of the feasibility of designing libraries in the set-based format and performing the conversion of the set-based definitions and results to the type-based format, thereby avoiding the use of the framework Types-To-Sets. Unfortunately, such an approach significantly undermines the benefits of the dedicated proof procedures designed or tailored for the type-based reasoning with the axiomatic type classes. The differences in the proof effort become apparent when comparing the compatible parts of the type-based standard library and the set-based library HOL-Algebra [4]. For example, because of the extensive use of the axiomatic type classes in the standard library, the rewrite rules typically do not need to carry additional assumptions, making the standard and efficient proof method simp (e.g., see [31]) based on conventional term rewriting [1] applicable for proving many of the common goals. Compare the statement of the set-based associativity law for a group (note that the presentation below is only an approximation)

\[
G(\otimes) \rightarrow \forall x_{\alpha}, y_{\alpha}, z_{\alpha} \in G_{\alpha} \text{set}. (x \otimes y) \otimes z = x \otimes (y \otimes z)
\]

from HOL-Algebra with its type-based counterpart

\[
\forall x_{\alpha_{\text{group}}}, y_{\alpha_{\text{group}}}, z_{\alpha_{\text{group}}}. (x \otimes y) \otimes z = x \otimes (y \otimes z)
\]

from the standard library. Whenever the set-based theorem above is used as a rewrite rule, the side conditions (such as \( x_{\alpha} \in G \)) associated with it need to be resolved. On the other hand, the use of the type-based formulation in conjunction with the axiomatic type classes allows viewing the associativity law as a part of a conventional term rewrite system [1]: even the assumption that indicates that \((\alpha_{\text{group}}, \otimes)\) is a group is implicit.

Returning to our example, potentially, the framework Types-To-Sets allows for the automated conversion of the theorem

\[
\forall A_{\text{tn}, \text{set}}. B_{\text{tn}, \text{set}}. \text{cl} A \rightarrow \text{cl} B \rightarrow \text{cl} (A \cup B)
\]

given by Eq. 2 to a theorem similar to the one given by Eq. 3:

\[
\begin{align*}
\text{ts}_{\text{with}} U \tau & \rightarrow \forall A_{\alpha}, B_{\alpha} \text{ set} \subseteq U. \\
\text{cl}_{\text{with}} U \tau A & \rightarrow \text{cl}_{\text{with}} U \tau B \rightarrow \text{cl}_{\text{with}} U \tau (A \cup B).
\end{align*}
\]

This is achieved via the application of the so-called relativization algorithm (RA) associated with Types-To-Sets. The left columns in Tables 1 and 2 show a significant part of the Isabelle/Isar code that was used for the conversion of the type-based theorem given by Eq. 2 and stated as closed_un to its set-based form similar to the theorem given Eq. 3 and stated as sb_ne_closed_un. The code listing demonstrates the state-of-the-art automation that was employed in [16].

The first several steps of the relativization algorithm, effectively, "unoverload" the type-based theorem given by Eq. 2, yielding the theorem given by Eq. 1, which we restate for convenience:

\[
\begin{align*}
\text{ts}_{\text{with}} \tau & \rightarrow \forall A_{\alpha} \text{ set}, B_{\alpha} \text{ set}. \\
\text{cl}_{\text{with}} \tau A & \rightarrow \text{cl}_{\text{with}} \tau B \rightarrow \text{cl}_{\text{with}} \tau (A \cup B).
\end{align*}
\]

Of course, the unoverloaded constants such as \( \text{cl}_{\text{with}} \) can be used repeatedly and, therefore, they are meant to be obtained before the application of the relativization algorithm as part of a setup for its application. A constant like \( \text{ts}_{\text{with}} \) is provided automatically during the specification of the class \( ts \) (such constants are known as the locale predicates [2, 9]), but the constant \( \text{cl}_{\text{with}} \) needs to be introduced manually (see the definition of the constant closed_with in the code listing in the left column of Table 1). Furthermore, the users are responsible for establishing an adequate relationship between the constants \( \text{cl} \) and \( \text{cl}_{\text{with}} \) (i.e., \( \text{cl} = \text{cl}_{\text{with}} \) open stated as the lemma closed_with in the code listing in the left column of Table 1).

The users also need to perform the relativization of the constants that occur in the theorem given by Eq. 2, that is, provide the transfer rules for these constants that are suitable for the application of the relativization algorithm. The aforementioned transfer rules are associated with the framework Transfer of Isabelle/HOL [11]. The framework Transfer allows for the transfer of theorems from one type to another [11]. For example, given a theorem \( \forall x_{\text{nat}} \geq 0, x < x + 1 \) about the type int of integers, the framework Transfer allows for its semi-automated transfer to the type nat of natural numbers yielding a theorem like \( \forall n_{\text{nat}}, n < n + 1 \) (in general, the framework allows for bidirectional transfer under suitable conditions) [11]. The automation is enabled by the transfer rules that need to be available for every constant that occurs in the theorem to which the framework Transfer needs to be applied. Fundamentally, the transfer rules are theorems
An Extension of the Framework Types-To-Sets for Isabelle/HOL

Table 1. Comparison of the original interface of Types-To-Sets and the CTR/ETTS: definitions

<table>
<thead>
<tr>
<th>Original Interface of Types-To-Sets</th>
<th>CTR/ETTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>definition closed_with :: (α set → B) → α set → B</td>
<td></td>
</tr>
<tr>
<td>where closed_with τ S = τ (¬S)</td>
<td></td>
</tr>
<tr>
<td>lemma closed_with: closed = closed_with open</td>
<td>ud topological_space.closed</td>
</tr>
<tr>
<td>unfolding closed_def closed_with_def .</td>
<td>ud closed’ closed</td>
</tr>
<tr>
<td>definition closed_on_with :: α set → (α set → B) → α set → B</td>
<td>ctr relativization</td>
</tr>
<tr>
<td>where closed_on_with U τ S = τ (¬S ∩ U)</td>
<td>synthesis ctr_simps</td>
</tr>
<tr>
<td>lemma closed_with.transfer[transfer_rule]: assumes [transfer_domain_rule]: Domainp A = (λx. x ∈ U) and [transfer_rule]: right_total A and [transfer_rule]: bi_unique A</td>
<td>assumes [transfer_domain_rule]: Domainp A = (λx. x ∈ U) and [transfer_rule]: right_total A and [transfer_rule]: bi_unique A</td>
</tr>
<tr>
<td>shows (rel_set A ===&gt; (=)) ===&gt; rel_set A ===&gt; (=) (closed_on_with U) closed_with</td>
<td>trp (‘a A)</td>
</tr>
<tr>
<td>(closed_on_with_def closed_with_def apply transfer_prover_start apply transfer_step+ by auto)</td>
<td>in closed_on_with: closed.with_def</td>
</tr>
</tbody>
</table>

Table 2. Comparison of the original interface of Types-To-Sets and the CTR/ETTS: theorems

<table>
<thead>
<tr>
<th>Original Interface of Types-To-Sets</th>
<th>CTR/ETTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>tts_context</td>
</tr>
<tr>
<td>assumes ltd:</td>
<td>tts: (?a to U)</td>
</tr>
<tr>
<td>∃Repβ→α Absα→β. type_definition Rep Abs U</td>
<td>rewriting ctr_simps</td>
</tr>
<tr>
<td>begin</td>
<td>substituting</td>
</tr>
<tr>
<td>interpretation local_typedef U TYPE(β)</td>
<td>topological_space_on_with_axioms</td>
</tr>
<tr>
<td>by unfold_locales (rule ltd)</td>
<td>eliminating through simp</td>
</tr>
<tr>
<td>lemmas_with [</td>
<td>begin</td>
</tr>
<tr>
<td>unfolded closed_with,</td>
<td>tts_lemmas in sb_ne_closed_Un = closed_Un</td>
</tr>
<tr>
<td>unoverloaded_type α,</td>
<td>end</td>
</tr>
<tr>
<td>where αβ,</td>
<td>lemmas sb_ne_closed_Un = sb_ne_closed_Un’</td>
</tr>
<tr>
<td>untransferred,</td>
<td>[</td>
</tr>
<tr>
<td>unfolded Collect_mem_eq,</td>
<td>cancel_type_definition,</td>
</tr>
<tr>
<td>rotated 2,</td>
<td>simplified subset_simp</td>
</tr>
<tr>
<td>0 topological_space_on_with_axioms</td>
<td>]</td>
</tr>
<tr>
<td>]: sb_ne_closed_Un’ = closed_Un</td>
<td></td>
</tr>
</tbody>
</table>

of the form R t t’, where Rα→β→B is referred to as the parametricity relation, tα and t’α are terms (usually constants or constant applications) and the types σ and τ have the same form α κ (in what follows the overbar notation ̄x will denote sequences), where κ is a function (written using postfix notation) that contains the information about the type constructors [21]. The parametricity relation must obey certain rules outlined, for example, in [21], but the details are not important at this stage. Also, practically, the parametricity relation may depend on additional parameters and transfer rules may have additional side conditions [21]. For example, a suitable transfer rule for the constant ts_with could look like

\[ R \left[ A_{α→β→B} \right] (ts_{\text{with}}^{\text{on}} U_{α set}) \] ts_with, \]

where

\[ R [A] = ((\text{rel_set} A \Rightarrow (=)) \Rightarrow (=)) \]

and the constants \( \Rightarrow \) (applied using the infix notation) and rel_set are known as a relators (relators lift relations over type constructors and constitute the backbone of the parametricity relations [21]). The rule holds under the side conditions Domainp A = (λx. x ∈ U), right_total A and bi_unique A. This is consistent with the restrictions imposed
on the admissible transfer rules for Types-To-Sets: the parameters of the relators of the transfer rules must be binary relations that are at least bi-unique (i.e., left-unique and right-unique) and right-total based on the default order of predicates in Isabelle/HOL [22]. The constant \( ts^m \) with is referred to as the relativization of the constant \( ts^m \) with. While the implementation of the relativization of the constant \( ts^m \) with is omitted from the left column of Table 1 for conciseness, the implementation of the relativization of the constant \( cl \) with is shown. From the code listing, it can be seen that the Transfer infrastructure already provides a certain level of automation for proving transfer rules [21]. Nonetheless, the statement of the set-based definitions and the transfer rules for the type-based definitions remain at the discretion of the users.

After completing the setup shown in the left column of Table 1, we apply the relativization algorithm associated with Types-To-Sets. The code listing in the left column of Table 2 shows the steps that the users would normally need to take for the conversion of the theorem given by Eq. 2 (\( \text{sb}_{\text{ne}} \text{closed}_\text{Un} \)) to a theorem similar to the one given by Eq. 3 (\( \text{sb}_{\text{ne}} \text{closed}_\text{Un} \)). A detailed description of the relativization algorithm is postponed until Section 4.

### 3.4 Extension of Types-To-Sets: Purpose and Scope

The experience of the early adopters of the framework Types-To-Sets [16] suggests that almost any significant practical application of Types-To-Sets requires an in-depth understanding of the relativization algorithm and a low-level interaction with a vast array of advanced features of the system (e.g., axiomatic type classes [9, 32, 40], ad hoc overloaded constant definitions [23, 40], locales [2, 3, 5, 17], Transfer [11]). Even if all of these prerequisites are fulfilled, arguably, the procedure for the application of the algorithm is still cumbersome and results in a significant amount of menial effort and boilerplate code, as can be inferred from the descriptions of the code listings in Subsection 3.3.

Both in [24] and [16], the authors call for the implementation of the infrastructure that would “streamline” the relativization process. More specifically, in [16], the authors make a suggestion that the automation should be presented in the form of an extension of the language Isabelle/Isar. In this article, we showcase our proposal for such an extension of the language and an associated software framework. In what follows, this extension and the associated software will be referred to as the Extension of Types-To-Sets (ETTS). We also provide an additional auxiliary software framework Conditional Transfer Rule (CTR) for semi-automated unoverloading and relativization of type-based constants.

The framework CTR provides the command \( \text{ud} \) for unoverloading of constants, and the command \( \text{ctr} \) for the synthesis of conditional transfer rules and relativization of constants. The right column of Table 1 showcases the application of these commands to unoverloading and relativization of the constant \( cl \). Unoverloading and relativization no longer require either explicit statements of definitions or invocation of any proof methods (cf. the code listing in the left column of Table 1). While the commands are not universally applicable, they can still save a significant amount of menial effort and many lines of boilerplate code. We provide a more detailed description of these commands in Section 5, but note that the underlying algorithms were designed before our work on the CTR and some of them were implemented and used in other contexts (e.g., see [8, 15, 25]).

After completing the setup shown in the right column of Table 1, we apply the relativization algorithm associated with Types-To-Sets to the theorem given by Eq. 2 via the interface of the ETTS using the code listing in the right column of Table 2. The invocation of the command \( \text{tts\_context} \) in the code listing is used for the parameterization of the relativization algorithm. Once the parameterization is completed, the command \( \text{tts\_lemmas} \) is used for the application of the relativization algorithm to the theorem \( \text{closed}_\text{Un} \). While the parameterization of the relativization algorithm shown in the code listing may still seem nontrivial, all keywords apart from the keyword \( \text{tts} \) are used only for post-processing of the raw output of the relativization algorithm. Thus, the invocation of

\[
\text{tts\_context tts: (?a to U)} \\
\text{begin} \\
\text{tts\_lemmas in sb\_closed\_Un = closed\_Un} \\
\text{end}
\]

suffices for the application of the raw relativization algorithm, similar to the one proposed in [22, 24]. In this case, the effort required from the users for the parameterization of the relativization algorithm is limited to the specification of the isomorphic type-set pairs of the form \((\alpha, U \cup \beta)\).

For now, we avoid a detailed description of other parameters of the relativization algorithm used in the code listing in the right column of Table 2. Nonetheless, we mention that the relativization algorithm may introduce redundant assumptions as part of its operation. While there is no simple general solution to this problem, the framework ETTS provides facilities for removing such redundant assumptions using the standard proof methods of Isabelle. In the example presented in Table 2, the raw output of the relativization algorithm contains the redundant assumption of the form \( U \neq \emptyset \), which we found difficult to remove using the existing technology in the code listing in the left column of the table. However, the ETTS removes this assumption using the standard proof method simp invoked as a consequence of the presence of the line \textbf{eliminating through simp} in the parameterization of the relativization algorithm.

We hope that the advantages of the ETTS are apparent. However, we also note that the example presented in this section does not showcase all of the features of the ETTS and the additional benefits that they can offer.
4 Previous Work

4.1 Background
In the remainder of this article, we provide a description of the implementation of the CTR and the ETTS, and showcase several application examples of these frameworks. However, before we proceed, we provide an account of the theory behind Types-To-Sets and its current implementation.

4.2 A Note on Conventions
From now on, we will try to be more careful in our use of notation, which is expected to be similar to the notation used in [22–24]. However, a disparity comes from our use of explicit notation for so-called schematic variables. In Isabelle/HOL, free variables that occur in the theorems at the top-level in the theory context are generalized implicitly, which may be expressed by replacing fixed variables by schematic variables [45]. In this article, the schematic variables will be prefixed with the question mark “?”, like so: ?a. Nonetheless, explicit quantification over the type variables at the top-level is also allowed: ∀x. 𝜙 [x]. Lastly, the square brackets may be used either for the denotation of substitution or to indicate that certain types/terms are a part of a term: 𝑡 [ ?a].

4.3 Relativization Algorithm
Let \( \alpha (\beta \approx U)_{\text{Abs}} \) denote
\[
(\forall x: x \in U) \land (\forall x: \text{Abs} \ (\text{Rep} \ x) = x) \land (\forall y: y \in U \rightarrow \text{Rep} \ (\text{Abs} \ y) = y)
\]
that can be interpreted as an isomorphism between a type \( \beta \) and a nonempty set \( U_{\alpha} \) exhibited by the mappings \( \text{Rep} \) and \( \text{Abs} \) [22], let \( \rightsquigarrow \) denote the constant/type dependency relation [22], let \( \rightsquigarrow^{1} \) be a substitutive closure of the constant/type dependency relation, let \( R^{*} \) denote the transitive closure of a binary relation \( R \), let \( \Delta_{c} \) denote the set of all types for which \( c \) is overloaded and let \( \sigma \not\in S \) mean that \( \sigma \) is not an instance of any type in \( S \) [22, 23].

The framework Types-To-Sets extends Isabelle/HOL with two axioms: Local Typedef Rule (LT) and the Unoverloading Rule (UO). The consistency of Isabelle/HOL augmented with the LT and the UO is proved in Theorem 11 in [23]. The LT is given by
\[
\Gamma \vdash U \neq \emptyset \quad \Gamma \vdash \left( \exists \text{Abs} \ \text{Rep.} \ \sigma (\beta \approx U)_{\text{Abs}} \rightarrow \phi \right) \quad \beta \notin U, \phi, \Gamma
\]
Thus, if \( \beta \) is fresh for the non-empty set \( U_{\sigma} \) set, the formula \( \phi \) and the context \( \Gamma \), then \( \phi \) can be proved in \( \Gamma \) by assuming the existence of a type \( \beta \) isomorphic to \( U \) [22]. The UO is given by
\[
\phi \quad \forall x: x \not\approx U_{\sigma} \rightarrow \exists u \in \phi. \neg(u \not\approx^{1} c_{a}) \land \sigma \not\in \Delta_{c}
\]
Thus, a constant-instance \( c_{a} \) may be replaced by a universally quantified variable \( x_{a} \) in \( \phi \) if all types and constant-instances in \( \phi \) do not depend on \( c_{a} \) through a chain of constant and type definitions and there is no definition for \( c_{a} \) [22]. The statement of the original relativization algorithm (ORA) can be found in Subsection 5.4 in [22]. Here, we present a variant of the algorithm that includes some of the amendments that were introduced in [16], which will be referred to as the relativization algorithm (RA). The differences between the ORA and the RA are largely implementation-specific.

Let \( Y \) be a class that depends on the overloaded constants \( \bar{c} \) and let \( U \downarrow \bar{f} \) be used to state that \( U \) is closed under the operations \( \bar{f} \); then the RA is shown in Figure 1. The input to the RA is assumed to be a theorem \( \vdash \phi [ ?\alpha ] \). Step 1 will be referred to as the first step of the dictionary construction (Subsection 5.2 in [22]); step 2 as unoverloading of the type \( ?\alpha \); it includes class internalization (Subsection 5.1 in [22]) and the application of the UO (it corresponds to the application of the attribute [44] unoverload_type [16]); step 3 provides the assumptions that are the prerequisites for the application of the LT; step 4 is reserved for the concrete type instantiation; step 5 is the application of Transfer (Section 6 in [22]); step 6 refers to the application of the LT; step 7 is the export of the theorem from the local context [43].

4.4 Implementation of Types-To-Sets
In [22], the authors extended the implementation of Isabelle/HOL with the LT and UO. Also, they introduced the attributes internalize_sort, unoverload and cancel_type_definition that allowed for the execution of steps 1, 3 and 7 (respectively) of the ORA. Other steps could be performed using the technology that already existed. In [16], the implementation was augmented with the attribute unoverload_type, which largely subsumed the functionality of the attributes internalize_sort and unoverload (the left column of Table 2 shows an application example).

The unoverloading and relativization of constants for the application of the RA was performed manually (e.g., see the left column of Table 1) in [16]. Nonetheless, unoverloading could be performed using the classical overloading elimination algorithm proposed in [20], but it is likely that an implementation of this algorithm was not publicly available at the time of writing. In [15], an alternative algorithm was made available via the command unoverload_definition, although it suffers from several limitations in comparison to the algorithm presented in [20] (see Subsection 5.2). The transfer rules for the constants that are conditionally parametric can be synthesized automatically using the command parametric_constant [8] from the standard distribution of Isabelle; the framework autoref [25] allows for the synthesis of transfer rules \( R t t' \), including both the parametricity relation \( R \) and the term \( t \), based on \( t' \), under favorable conditions; lastly, in [25] and [16], the authors suggest an outline of another feasible algorithm for the synthesis of the transfer rules based on the functionality of the framework Transfer [11], but do not provide an implementation.
5 Conditional Transfer Rule

5.1 Background

In this section, we describe the implementation and the functionality of the framework CTR and the associated Isabelle/Isar commands ud and ctr.

5.2 Unoverloading

In this subsection, we describe the command ud for unoverloading of constants. The command ud uses a restricted (non-recursive) variant of the classical unoverloading elimination algorithm. It is independent of the LT, UO and the type class infrastructure, unlike the command unoverload_definition. The algorithm is explained in detail in Appendix A. Here, we provide an outline based on an example similar to Example 5 from Subsection 5.2 in [20].

Consider the declaration pls_ab → α → α and its definitions pls_ab abol → α and

\[
\text{pls}_{\alpha \times \beta \rightarrow \alpha} = \lambda x y. (\text{pls}_{\text{fst}} (x)) (\text{fst} y), \text{pls}_{\text{snd}} (x) (\text{snd} y),
\]

where (+) is a standard operation on the type nat from the standard library, × (in infix form) denotes the type constructor for the canonical product type, while fst and snd are the associated projections [34].

Consider the invocation ud pls_ab pls_ab pls_ab pls_ab followed by ud pls_times pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pla

The axiom is instantiated by substituting nat for α. Afterwards, pls = pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pls_ab pla

The final steps of the algorithm establish the relationship between the overloaded and unoverloaded constants:

\[
\text{pls}_{\text{nat} \times \beta} = \text{pls}_{\text{nat} \times \beta} \text{ with } \text{pls}_{\beta} \rightarrow \beta.
\]

While the command is almost universally applicable in practice, it suffers from the limitation outlined in [20, 22, 24]; it cannot be applied to the constants whose types contain occurrences of the type constructors whose type definitions contain occurrences of unresolvable overloading.

5.3 Relativization of Constants

In this subsection, we describe the command ctr. The command offers two algorithms for the relativization of constants that have a distinct (but overlapping) scope of applicability. The first algorithm (CTR I) was proposed and implemented in [8]. An outline of the second algorithm (CTR II) was proposed in [25] and [16], but not implemented.

A description of CTR I can be found in [8] and will not be restated. A detailed description of CTR II is given in Appendix B. Here, we only present a brief outline. Generally, given a set of side conditions on A and a constant-instance definition c_A[β] = t, CTR II produces the parametricity relation R[A] for the type σ based on the algorithm from Subsection 4.1.1 in [21] and solves R[A] ? x for ? x using the method transfer_prover (Subsection 4.6 in [21]). A solution t′ : σ[α] does not exist for the definition of a new constant c_A[β] with such that c_A[α] for some variables x. It should be noted that neither CTR I nor CTR II can guarantee that they can identify whether a transfer rule exists for a given constant under a given set of side conditions, nor that it will be found if it exists.
6 Extension of Types-To-Sets

6.1 Background

In this section, we describe our implementation of the framework ETTS in Isabelle/ML that offers the integration of Types-To-Sets with the Isabelle/Isar infrastructure and automation of a variant of the relativization algorithm similar to the RA. The primary functionality of the ETTS is available via the Isabelle/Isar commands \texttt{tts\_context}, \texttt{tts\_lemmas} and \texttt{tts\_lemma}. The commands \texttt{tts\_lemmas} and \texttt{tts\_lemma}, when invoked inside an appropriately defined \texttt{tts\_context}, provide the functionality that is approximately equivalent to the application of the RA and several additional steps of pre-processing of the input and post-processing of the result (collectively referred to as the \textit{extended relativization algorithm} or ERA). There also exists a secondary command aimed at resolving certain problems that one may encounter during relativization: \texttt{tts\_register\_sbts}. More specifically, this command provides means for using transfer rules stated in a local context during the step of the ERA that is similar to step 5 (the application of Transfer) of the RA. The functionality of this command is explained in more detail in Subsection 6.3 below. The ETTS was implemented following conventional software engineering practices and partially tested using the framework SpecCheck \[18\].

6.2 Code Generation

The interface of the ETTS serves another important purpose that has not yet been mentioned in this article. The ERA (and, to a lesser extent, the RA) can behave in an unpredictable manner: the final result of the ERA depends on the transfer rules and various theorems that are used during post-processing. The aforementioned transfer rules and theorems may change throughout the course of time in any given body of work. This can cause problems with backward compatibility: it is not possible to infer the exact form of a synthesized theorem before the changes in the dependencies took place. Moreover, normally, the statements of the synthesized theorems do not appear in the automatically generated formal proof documents.

The solution that was adopted during the design of the interface of the ETTS was to, indeed, provide means for stating the synthesized theorems in the Isabelle/Isar language. This process was automated via the use of the frameworks for code generation and active areas: both frameworks are parts of the existing infrastructure of Isabelle/HOL (e.g., see \[27, 36\]).

Figure 2 showcases the workflow that is meant to be followed when using the ETTS. The command \texttt{tts\_lemmas} provides an interface for the invocation of the tools for code generation, whereas the command \texttt{tts\_lemma} serves a purpose similar to the standard command \texttt{lemma} (see \[41\] and \[42\]), but uses the output of the ERA to discharge the proof obligations.

6.3 Set-Based Terms and Their Registration

One of the most challenging aspects of the automation of the relativization process is related to the application of Transfer during step 5 of the RA: a suitable transfer rule for a given constant-instance may exist only under non-conventional side conditions (e.g., consider the Hilbert choice operator $\varepsilon$ \[16, 24\]). While the ETTS does not offer a fundamental solution to this problem, it does provide the additional infrastructure that may improve the user experience when dealing with the transfer rules that can only be conveniently stated in an explicitly relativized local context (usually a relativized locale): a common problem that was already explored in \[16\].

The users of the ETTS can use the command \texttt{tts\_register\_sbts} for the registration of the so-called \textit{set-based terms} (sbterms) in a so-called \textit{sbt-database}. Semantically, sbterms are relativizations of arbitrary (but compatible) type-based concepts. To each sbterm $t_{\alpha\beta}^{\gamma}$ in the local context $\Gamma$, there is associated a deduction of the form $\Gamma \vdash \text{dbr}[?\bar{A}, ?\bar{U}] \rightarrow \exists b. \bar{R} [?\bar{A}]_{\alpha \rightarrow ?\bar{b} \rightarrow ?\bar{\beta} \rightarrow ?\bar{\gamma} \rightarrow b}$, where $\Gamma$ is a parametricity relation and $\text{dbr}[\cdot, \cdot]$ is an abbreviation that represents the canonical set of assumptions on the parameters of the relators for the relativization algorithm (see Subsection 3.3).

For example, following Subsection 3.2, assuming that a local context $\Gamma$ contains the assumption $\text{ts}^{\text{on}}_{U \tau}$, we may wish to view $\tau$ as an sbterm. During the setup of the relativization algorithm in the context $\Gamma$, we could “associate” $\tau$ with the result of unoverloading of the constant open (e.g., a variable $\tau'$), allowing for the instantiation of the aforementioned deduction to yield a transfer rule of the form $\Gamma' \vdash R \tau \tau'$ in some context $\Gamma'$ such that $\Gamma \subseteq \Gamma'$ and $\tau' \in \Gamma'$.
6.4 ERA: Foundations and Parameterization

The ERA can be seen as a localized variant of the RA that includes certain implementation-specific details and provides several additional steps for pre- and post-processing.

In what follows, assume the existence of an underlying well-formed definitional theory [23] that contains all definitional axioms that appear in the standard library. If \( \Gamma \vdash \alpha \) (\( \approx \) U)\text{Rep} and \( \beta, U_{\alpha \text{ set}}, \text{Rep}_{\beta \rightarrow \alpha}, \text{Abs}_{\alpha \rightarrow \beta} \in \Gamma \), then the 4-tuple \( (U_{\alpha \text{ set}}, \beta, \text{Rep}_{\beta \rightarrow \alpha}, \text{Abs}_{\alpha \rightarrow \beta}) \), will be referred to as a relativization isomorphism (RI) with respect to \( \Gamma \) (or RI, if \( \Gamma \) can be inferred). Given the RI \( (U_{\alpha \text{ set}}, \beta, \text{Rep}_{\beta \rightarrow \alpha}, \text{Abs}_{\alpha \rightarrow \beta}) \), the term \( U_{\alpha \text{ set}} \) will be referred to as the set associated with the RI, \( \beta \) will be referred to as the type variable associated with the RI, \( \text{Rep}_{\beta \rightarrow \alpha} \) will be referred to as the representation associated with the RI, and \( \text{Abs}_{\alpha \rightarrow \beta} \) will be referred to as the abstraction associated with the RI. Moreover, any typed term variable \( T_{\gamma \rightarrow \alpha \rightarrow \beta} \) such that \( \Gamma \vdash T = (\lambda x. y. \mathcal{R} y = x) \) will be referred to as the transfer relation associated with the RI.

The parameterization of the ERA can be performed in an arbitrary context \( \Gamma \) (this parameterization will be referred to as the \text{ERA}-parameterization for \( \Gamma \)) using the command \textit{tts\_context} (an example is given in Table 2). The ERA is parameterized by the RI specification, the sbterm specification, the rewrite rules for the set-based theorem, the known premises for the set-based theorem, the specification of the elimination attributes for the RI, and the side conditions associated with the RI.

The \text{sbterm} specification is a finite non-empty sequence of pairs \( (?\gamma, U_{\alpha \text{ set}}) \), such that \(?\gamma\) is a schematic type variable and \( U_{\alpha \text{ set}} \in \Gamma \). The individual elements of the \text{RI} specification will be referred to as the \text{RI} specification elements. The first element of the \text{RI} specification element will be referred to as the schematic type variable associated with the \text{RI} specification element, the second element will be referred to as the set associated with the \text{RI} specification element.

The \text{sbterm} specification is a finite sequence of pairs \( (\tilde{\alpha}, K, u : \tilde{\beta} K) \), where \( \tilde{\alpha} \vdash K \) is either a constant-instance or a schematic typed term variable and \( u : \tilde{\beta} K \) is an sbterm with respect to \( \Gamma \). The notation for the elements of the \text{sbterm} specification follows a convention similar to the one introduced for the \text{RI} specification elements.

The rewrite rules for the set-based theorem can be any finite set of valid rules for the Isabelle simplifier [44]; the known premises for the set-based theorem can be any finite sequence of deductions in \( \Gamma \); the specification of the elimination of premises in the set-based theorem is a pair \( (t, m) \), where \( t \) is a finite sequence of formulae and \( m \) is a proof method; the attributes for the set-based theorem is a finite sequence of attributes of Isabelle.

6.5 Definition of the ERA

The relativization is performed inside a local context \( \Gamma \) with an associated ERA-parameterization (such as context-parameterization pair will be called a \textit{tts context}). The ERA provides explicit support for handling the transfer rules local to the context through the infrastructure for the registration of the sbterms (see Subsection 6.5). Apart from this, the main part of the ERA is similar to the RA. However, the ERA also provides several tools for post-processing of the raw result of the relativization. The ERA also has an initialization stage, but this stage is largely hidden from the end-user.

Thus, the ERA can be divided into three distinct parts: initialization of the relativization context, kernel of the ERA (KERA) and post-processing, each of which is described below. Assume that the context \( \Gamma \) contains the variable \( U_{\alpha \text{ set}} \) and the finite sequences of variables \( \tilde{g} \) and \( \tilde{f} \) indexed by \( I \) and \( J \), respectively, such that \( \tilde{g}_i : \alpha K_i \) and \( \tilde{f}_j : \alpha L_j \) for all \( i \in I \) and \( j \in J \) for some finite sequences of functions \( K \) and \( L \) representing the type constructors and also indexed by \( I \) and \( J \), respectively. Also, assume that the input to the ERA is the type-based theorem \( \Gamma \vdash \phi [?\alpha_1, \tilde{h}[?\alpha_1]] \) such that \( \tilde{h} \) is indexed by \( I \) and \( \Gamma \) depends on the overloaded constants \( \tilde{s} \) indexed by \( J \). Finally, assume that the ERA is parameterized by the RI specification \( (?\alpha_1, U_{\alpha \text{ set}}) \) and the \text{sbterm} specification elements \( (?\tilde{h}_i, \tilde{g}_i) \) and \( (?\tilde{f}_j, \tilde{f}_j) \) for all \( i \in I \) and \( j \in J \).

\textbf{Initialization of the relativization context.} Prior to the application of the relativization algorithm, the formula \( \exists \text{Rep Abs} ._{\alpha}(\beta \approx U)\text{Rep} \) is added to the context \( \Gamma \), with the type variable \( \beta \) being fresh for \( \Gamma \), resulting in a new context \( \Gamma' \) such that \( \Gamma \subseteq \Gamma' \) and \( \exists \text{Rep Abs} ._{\alpha}(\beta \approx U)\text{Rep} \in \Gamma' \).

Then, the properties of the Hilbert choice \( \varepsilon \) are used for the definition of \( \text{Rep} \) and \( \text{Abs} \) such that \( \Gamma' \vdash \alpha(\beta \approx U)\text{Rep} \). Finally, the transfer domain rule associated with the RI and the side conditions associated with the RI are proved for \( \Gamma' \) with respect to \( \Gamma' \). Furthermore, a fresh \( T_{\alpha \rightarrow \beta \rightarrow \beta} \) (for \( \Gamma' \)) is defined as a transfer relation associated with the RI. Finally, the transfer domain rule associated with the RI and the side conditions associated with the RI are proved for \( \Gamma' \) with respect to \( \Gamma' \). For each \( \tilde{g}_i \), such that \( i \in I \), the sbt-database contains a deduction \( \Gamma' \vdash \text{dbr}[?A, U] \rightarrow \exists a. R[?A]_\alpha K_i \rightarrow \gamma \tilde{g}_i \). Hence, for each \( i \in I \), \( \tilde{g}_i \) is instantiated as \( \gamma \tilde{g}_i \) for \( \gamma \tilde{g}_i \) and \( \tilde{g}_i \) for \( \tilde{g}_i \). Then, \( \tilde{g}_i \) is obtained via \( \text{Rep}_{\alpha \rightarrow \beta \rightarrow \beta} \) and \( \tilde{g}_i \) for \( \tilde{g}_i \). Similar deductions are also established for the sequence \( \tilde{f}_j \), with the sequence of the variables appearing on the right-hand side of the transfer rules denoted by \( b \). These deductions are meant to be used by Transfer during the step of the KERA that is equivalent to step 5 of the RA, as shown below.

\textbf{Kernel of ERA.} The KERA is similar to the RA and shown in Figure 3. Thus, step 1 will be referred to as the first step of the dictionary construction (similar to step 1 of the RA); step 2 will be referred to as unloading of the type \(?\alpha_1\); it includes class internalization and the application of the
UO (similar to step 2 of the RA); in step 3, ?α is instantiated as β using the RI specification (similar to step 4 in the RA); in step 4, the sbterm specification is used for the instantiation of ?h as α and ?f as b; step 5 refers to the application of Transfer, including the transfer rules associated with the sbterms (similar to step 5 in the RA); in step 6, the result is exported from Γ′ to Γ, providing the additional premise ∃Rep Abs. α (β ≈ U) Abs; step 7 is the application of the attribute cancel_type_definition (similar to step 6 in the RA). The RI specification and the sbterm specification provide the information that is necessary to perform the type and term substitutions in steps 3 and 4 of the KERA. If the specifications are viewed as finite maps, their domains morph along the transformations that the theorem undergoes until step 4.

Post-processing. The deduction that is obtained in the final step of the KERA can often be simplified further. The following post-processing steps were created to allow for the presentation of the set-based theorem in a format that is both desirable and convenient for the usual applications:

1. Simplification. The rewriting is performed using the rewrite rules for the set-based theorem, relying on the functionality of the standard simplifier of Isabelle.
2. Substitution of known premises. The known premises for the set-based theorem are matched with the premises of the set-based theorem, allowing for their elimination.
3. Elimination of premises. Each premise is matched against each term in the specification of the elimination of premises in the set-based theorem; the associated method is applied in an attempt to eliminate the matching premises (e.g., this can be useful for the elimination of the premises of the form U ≠ ∅).
4. Application of the attributes for the set-based theorem. The attributes for the set-based theorem are applied as the final step during post-processing.

Generally, the desired form of the result after the successful application of post-processing normally has a form similar to Γ′ ⊢ φ with [α, U, g, f] with the premises U ≠ ∅, U ⊢ g, f and Y on with U f eliminated completely.

6.6 ETTS and ERA by Example

We consider an example of the application of the ERA. We put an emphasis on the description of the features that were added to the Types-To-Sets infrastructure as part of our work on the ETTS, but we omit certain implementation-specific details and a description of the infrastructure for the registration of the sbterms.

We return to the example presented in the right column of Table 2. Thence, we consider the task of the relativization of the theorem closed_Un from the standard library (this time, the schematic variables are treated explicitly): + cl ?A ≠ set ⊢ cl ?B ≠ set → cl (?A ∪ ?B). We assume the existence of the locale topological_space_on_with with the corresponding locale predicate ts on with. We also assume that the constant cl was unoverloaded, resulting in the declaration of the constant cl with defined as + cl with = λ τ S. τ (~S) and the related theorem + cl = cl with.open. Furthermore, we assume the existence of the conditional transfer rules suitable for the application of the relativization algorithm for every constant that occurs in the theorem, including R ts [A] (ts on with U) τ and R cl [A] (cl on with U) cl with that hold under suitable side conditions on any admissible binary relation A and set U.

The relativization of the theorem takes place in the context Γ of the locale topological_space_on_with such that U α set, τ α set → U ⊢ Γ ⊢ ts on with U τ (this deduction is stored as topological_space_on_with_axioms).

The parameterization of the ERA is performed using the command tts_context, as shown in the right column of Table 2. For convenience of the reader, we restate the main part of the parameterization below:

tts_context

tts: (?α to U)
rewriting ctr_simps
substituting topological_space_on_with_axioms
eliminating through simp

...
Thus, \texttt{tts} is used for the definition of the RI specification element (?\texttt{ts}, U); \texttt{rewriting} is used for the definition of the rewrite rules for the set-based theorem, \texttt{substituting} is used for the definition of the known premises for the set-based theorem and \texttt{eliminating through} for the definition of the specification of the elimination of premises in the set-based theorem.

The relativization is performed using the invocation of the command \texttt{tts_lemmas} inside the \texttt{tts_context}:

\begin{verbatim}
  \texttt{tts_lemmas in sb\_closed\_Un = closed\_Un.}
\end{verbatim}

The right-hand side of the invocation is a reference to the type-based theorem and the left-hand side is the desired name for the result of the relativization. The invocation of the command \texttt{tts_lemmas} above initiates the execution of the ERA, as described in Subsection 6.5.

During the first part of the ERA, the initialization of the relativization context, the command builds the relativization context $\Gamma'$ starting from $\Gamma$ incrementally. At the end of this initialization, the context $\Gamma'$ is given by $\Gamma' = \Gamma \cup S$, where $S$ is a finite set such that $a (\beta \approx U)_{\text{Abs}} \cap \text{Rep}_{\beta \rightarrow \alpha}, \text{Abs}_{\alpha \rightarrow \beta}, T_{\alpha \rightarrow \beta \rightarrow B} \in \Gamma'$. The fixed variables that appear in the list above are related through the following deductions: $\Gamma' \vdash T = (\lambda \alpha \ x. r = \text{Rep} \ a)$, $\Gamma' \vdash \text{Domain} \ T = (\lambda x. x \in U)$, $\Gamma' \vdash \text{bi\_unique} \ T$ and $\Gamma' \vdash \text{right\_total} \ T$. Thus, implicitly, the relativization context provides the transfer relation and the requisite transfer rules for the application of the ERA.

The next stage of the ERA is the application of the KERA. We start with the deduction

$$\Gamma' \vdash \text{cl} \ ?A_{\text{Abs}}, \text{set} \rightarrow \text{cl} \ ?B_{\text{Abs}}, \text{set} \rightarrow \text{cl} \ (?A \cup ?B) .$$

We unfold cl = cl with open and unoverload using the UO:

$$\Gamma' \vdash \text{ts} \ ? \rightarrow \text{cl} \ ?A_{\text{Abs}}, \text{set} \rightarrow \text{cl} \ ?B_{\text{Abs}}, \text{set} \rightarrow \text{cl} \ ?A \cup ?B .$$

Next, we substitute the type variable $\beta$ for $\alpha$

$$\Gamma' \vdash \text{ts} \ ? \rightarrow \text{cl} \ ?A_{\text{Abs}}, \text{set} \rightarrow \text{cl} \ ?B_{\text{Abs}}, \text{set} \rightarrow \text{cl} \ ?A \cup ?B .$$

and apply Transfer:

$$\Gamma' \vdash \text{ts} \ ? \rightarrow \text{cl} \ ?A \cup ?B .$$

Finally, we export the deduction back to the context $\Gamma$

$$\Gamma \vdash \exists \text{Rep}, \text{Abs} \ _{\alpha} (\beta \approx U)_{\text{Abs}} \rightarrow \text{ts} \ ? \rightarrow \text{cl} \ ?A \cup ?B .$$
7 Application Examples

7.1 Introduction

There has been a growing interest in the use of Types-To-Sets for formalization (for example, see [7], [26] and [16]). Arguably, the most significant application example was developed in [16], which provides the relativization of over 200 theorems from the standard mathematics library of Isabelle/HOL. In this section, we mention several application examples that we developed as part of our work on the ETTS. More specifically, we describe the following three miniature libraries of formalized mathematics: TTS Vector Spaces, SML Relativization and TTS Foundations.

7.2 TTS Vector Spaces

The library TTS Vector Spaces is a remake of the library of relativized results that was developed in [16] using the ETTS. A description of the original library has already been given in [16] and will not be restated. The advantages of the ETTS are apparent: the complex infrastructure that was needed for compiling out dependencies on overloaded constants, the manual invocation of the attributes for the individual steps of the relativization algorithm and the repeated references to the theorem as it undergoes the transformations associated with the steps of the relativization algorithm were no longer needed. Furthermore, the theorems synthesized by the ETTS appear in the formal proof documents in a format that is similar to the canonical format of the Isabelle/Isar declarations associated with the standard commands, such as the command lemma.

The examples shown in Tables 1 and 2 are representative of the improvements that were achieved using the ETTS. However, to elaborate, we consider a more specific example. We choose the type-based theorem dim_sums_Int, as it was already used in a similar context in [16]. The type-based theorem is stated and proved in the standard library of Isabelle/HOL:

proposition dim_sums_Int:
assumes subspace S and subspace T
shows dim {x + y | x y. x ∈ S ∧ y ∈ T} + dim (S ∩ T) = dim S + dim T
proof –
obtain B where B: B ⊆ S ∩ T and S ∩ T ⊆ span B
…
qed

The proof of this type-based theorem consists of nearly 100 lines of Isabelle/Isar code. Of course, due to the arguments presented in Subsection 3.3, it is expected that the proof of its set-based counterpart would be significantly longer and less efficient. However, the relativization of this theorem using the ETTS in TTS Vector Spaces takes the following form:

tts_lemma dim_sums_Int:
assumes S ⊆ U and T ⊆ U
and subspace S and subspace T
shows dim {x ∈ U. ∃y ∈ U. ∃z ∈ U. x = y + z ∧ y ∈ S ∧ z ∈ T} + dim (S ∩ T) = dim S + dim T
is finite_dimensional_vector_space.dim_sums_Int.

Moreover, the statement and the proof of the set-based theorem above was generated automatically using a single line of Isabelle/Isar code:

tts_lemmas in
finite_dimensional_vector_space.dim_sums_Int

Of course, as explained in Subsection 6.5, the relativization can only take place in an appropriately parameterized tts context. Therefore, some setup was required before such seamless relativization could become possible. We present the code that was used for the parameterization of the tts context used for the relativization of the theorem dim_sums_Int below:

tts_context

tts: (?β to ) to set
sbterms: ( (+)β β, group_add→ +β } β to (+)β +β →β)
and ( )β group_add→ _β to ( _ 0)β
and ( β, group_add→ _β to _β +β)
and ( β, group_add→ _β to _β +β)
rewriting ctr_simps
substituting ab_group_add_ow_axioms
and vector_space_ow_axioms
and ab V.
finite_dimensional_vector_space_ow_axioms
and module_ow_axioms
applying
[ OFF
implicit_V.carrier_ne
implicit_V.minus_closed
implicit_V.minus_closed
basis_subset,
unfolded tts_implicit
]

While the code above may look slightly intimidating, most of it is dedicated to post-processing of the result of the KERA and the same setup can often be reused for the relativization of other similar theorems. In fact, only 11 invocations of the command tts_context were required in the library TTS Vector Spaces, yielding a ratio of the invocation of the command tts_context to set-based theorems less than 1 : 18.
The library SML Relativization provides a relativization of several further elements of the standard library. The relativization is performed largely in the spirit of the methodology that was showcased in Subsection 3.4. This methodology was used for the relativization of over 800 theorems in the areas of order theory, abstract algebra and general topology. Arguably, this library forms the largest application of the framework Types-To-Sets at the time of writing.

For completeness, we present an example of a relativized theorem from the library SML Relativization. The theorem connected_Times from the standard library showcases that the product of connected topological spaces is a connected topological space:

\begin{verbatim}
proposition connected_Times:
  assumes S: connected S and T: connected T
  shows connected (S × T)
proof-
  ... qed
\end{verbatim}

The relativization of this theorem in the library SML Relativization takes the following form:

\begin{verbatim}
tts_context
  tts: (?α to U₁) and (?β to U₂)
rewriting ctr_simps
substituting ts_1.topological_space_ow_axioms
  and ts_2.topological_space_ow_axioms
eliminating ?U ≠ 0 through
  (fold tts_implicit, unfold connected_ow_def, simp)
  applying [folded tts_implicit]
begin
  tts_lemma connected_Times:
    assumes S ⊆ U₁ and T ⊆ U₂
    and ts₁.connected S and ts₂.connected T
    shows connected (S × T)
    is connected_Times.
end
\end{verbatim}

\section{SML Relativization}

The commands \texttt{unoverload_definition} and \texttt{ud} have a distinct scope of applicability: the functionality of the commands can be combined. The command \texttt{ctr} can be improved by implementing additional algorithms for the synthesis of conditional transfer rules (e.g., see Acknowledgments).

The ETTS was designed under a policy of non-intervention with the implementation of Isabelle. However, it is possible to provide access to the primary functionality of ETTS via the standard commands \texttt{context}, \texttt{lemmas} and \texttt{lemma}. The registration of the sbterms can be integrated within the implementation of the commands \texttt{setup_lifting} and \texttt{lift_definition} [11].

Lastly, recently, formal verification of the consistency of HOL with ad hoc overloading was performed in [37], bringing the possibility of a mechanized verification of the consistency of Isabelle/HOL augmented with the LT and UO one step closer to reality.

\section{Acknowledgments}

The author would like to acknowledge the assistance received from the users of the mailing list of Isabelle [14] and the Stack Exchange network [38]. The author would also like to acknowledge the authors of [22] and [15] for their explanations of the functionality of Types-To-Sets and for providing some of the ideas that laid the foundation of this work. Special thanks go to Fabian Immler for the conceptual design of the commands \texttt{tts_context}, \texttt{tts_lemmas}, \texttt{tts_lemma} and for suggesting an outline of an algorithm for the relativization of constants (CTR II); to Andrei Popescu and three anonymous referees for providing constructive critical feedback and for the evaluation of this article; to Alexander Krauss for the assistance in the explanation of certain aspects of [20]; to Andreas Lochbihler for providing an outline of an improved algorithm for the relativization of constants (not currently implemented); to Andreas Lochbihler and Dmitriy Traytel for explaining the functionality of the framework \textit{Conditional Parametricity}. The author would also like to acknowledge the positive impact of [39] and [43] on his ability to code in Isabelle/ML. Lastly, the author would like to express gratitude to all members of his family and friends for their continuous support.
A Unoverloading in Detail

The general references for this subsection are [20] and [23]. The command ud relies on a restricted variant of the classical overloading elimination algorithm [20]. It is assumed that there exists a variable ud with that stores theorems of the form \( c = c_{\text{with}} \ast \), where \( c \) and \( c_{\text{with}} \) are distinct constant-instances and \( \ast \) is a finite sequence of uninterpreted constant-instances, such that, if \( c \) depends on a type variable \( \alpha \), with \( \alpha \) being a class that depends on the overloaded constants \( \ast' \), then \( \ast \) contains \( \ast' \) as a subsequence. Lastly, the binary operation \( \cup \) is defined in a manner such that for any sequences \( \ast \) and \( \ast' \), \( \ast \cup \ast' \) is a sequence that consists of all elements of the union of the elements of \( \ast \) and \( \ast' \) without duplication. Assuming an underlying well-formed definitional theory \( D \), the input to the algorithm is a constant-instance \( c_\ast \). Given the constant-instance \( c_\ast \), there exists at most one definitional axiom \( c_\ast = \phi_\ast [\ast] \) in \( D \) such that \( c_\ast \leq c_\ast \); otherwise the orthogonality of \( D \) and, therefore, the well-formedness of \( D \) are violated (\( \phi_\ast \) is assumed to be parameterized by the types that it can have with respect to the type substitution operation, and \( \ast \) in \( c_\ast = \phi_\ast [\ast] \) is a list of all uninterpreted constant-instances that occur in \( \phi_\ast [\ast] \)).

If a definitional axiom \( c_\ast = \phi_\ast [\ast] \) such that \( c_\ast \leq c_\ast \) exists for the constant-instance \( c_\ast \), then the following derivation is applied to it by the algorithm

\[
\frac{1}{\vdash c_\ast = \phi_\ast [\ast]} \quad (1) \\
\frac{\vdash c_\ast = \phi_\ast [\ast]}{\vdash c_\ast = \phi_{\text{with}} [\ast \cup \ast']} \quad (2) \\
\frac{\vdash c_\ast = \phi_{\text{with}} \{f \} \phi_{\text{with}} [\ast \cup \ast']} {\vdash c_\ast = \phi_{\text{with}} [\ast \cup \ast']} \quad (3) \\
\frac{\vdash c_\ast = \phi_{\text{with}} [\ast \cup \ast']} {\vdash c_\ast = \phi_{\text{with}} [\ast \cup \ast']} \quad (4) \\
\frac{\vdash c_\ast = c_{\text{with}} [\ast \cup \ast']} \quad (6)
\]

In step 1, the previously established property \( c_\ast \leq c_\ast \) is used to create the (extended variant of the) type substitution map \( \rho \) such that \( \sigma = \rho (\tau) \) (see [21]) and perform the type substitution in \( c_\ast = \phi_\ast [\ast] \) to obtain \( c_\ast = \phi_\ast [\ast] \); in step 2, the collection of theorems ud with is unfolded, using it as a term rewriting system, possibly introducing further uninterpreted constants \( \ast' \); in step 3, the term on the right-hand side of the theorem is processed by removing the sort constraints from all type variables that occur in it, replacing every uninterpreted constant-instance (this excludes all built-in constants of Isabelle/HOL) that occurs in it by a fresh term variable, and applying the abstraction until the resulting term is closed: this term forms the right-hand side of a new definitional axiom of a fresh constant \( c_{\text{with}} \) (if the conditions associated with the definitional principles of Isabelle/HOL are satisfied); step 4 is justified by the beta-contraction; step 5 is a substitution of the uninterpreted constants \( \ast \cup \ast' \); step 6 follows trivially from the results of the application of steps 2 and 5.

B CTR II in Detail

Assume the existence of an underlying well-formed definitional theory \( D \) and a context \( \Gamma \); assume the existence of a map \( c_{\text{rel}} \) from a finite set of non-nullary type constructors to relations, mapping each non-nullary type constructor in its domain to a valid relator for this type constructor. The inputs to CTR II are a constant-instance definition \( \vdash c_{\gamma K} = \phi [\gamma] \) of the constant-instance \( c_{\gamma K} \) and the map \( \text{trp} \) from the set of all schematic type variables in \( \gamma \) to the set of binary relations \( A_{\alpha \rightarrow \beta \rightarrow \gamma} \) in \( \Gamma \) with non-overlapping type variables \( \gamma \). \( \gamma \) corresponds to a sequence of all distinct type variables that occur in the type \( \gamma \), while \( K \), applied using a postfix notation, contains all information about the type constructors of the type \( \gamma \), such that the non-nullary type constructors associated with \( K \) form a subset of the domain of \( c_{\text{rel}} \). CTR II consists of three parts: synthesis of a parametricity relation, synthesis of a transfer rule and post-processing.

Synthesis of a parametricity relation. An outline of an algorithm for the conversion of a type to a parametricity relation is given in Subsection 4.1.1 in [21]. Thus, every nullary type constructor in \( \gamma \) is replaced by the identity relation =, every non-nullary type constructor \( k \) in \( \gamma \) is replaced by its corresponding relator \( c_{\text{rel}} (k) \) and every type variable \( \gamma \) in \( \gamma \) is replaced by \( \text{trp} (\gamma) \), obtaining the parametricity relation \( R_{\delta} K \rightarrow \delta K \rightarrow \beta \rightarrow \gamma \).

Synthesis of a transfer rule. First, the goal \( R \phi [\alpha] \phi [\beta] \) is created and an attempt to prove it is made using the algorithm associated with the method \( \text{transfer\_prover} \) in \( \Gamma \). If the proof is successful, nothing else needs to be done in this part. Otherwise, an attempt to find a solution for \( ? \) in \( R (\alpha_a K) \phi [\beta] \) is made, once again, using the \( \text{transfer\_prover} \). The result of the successful completion of the synthesis is a transfer rule \( \Gamma \vdash \text{R} \psi [\alpha, x] \phi [\beta] \), where \( x \) is used to denote a sequence of typed variables, each of which occurs free in the context \( \Gamma \) (the success of this part is not guaranteed).

Post-processing. If \( \psi [\alpha, x] = \phi [\alpha] \) after the successful completion of part 2 of the algorithm, then the definitions of the constant-instances \( c_{\alpha K} \) and \( c_{\beta K} \) are folded, resulting in the deduction \( \Gamma \vdash R c_{\alpha K} c_{\beta K} \). Otherwise, if \( \psi [\alpha, x] \neq \phi [\alpha] \), then a new constant \( d \) is declared such that \( \vdash d_{\psi [\alpha, x]} = (\lambda x. \psi [\alpha, x]) \), where \( \sigma \) is determined uniquely by \( x \) and \( ? \alpha K \). In this case, \( \Gamma \vdash R \psi [\alpha, x] \phi [\beta] \) can be restated as \( \Gamma \vdash R (d_{\psi [\alpha, x]}) c_{\beta K} \). This result can be exported to the global theory context and forms a conditional transfer rule for \( c_{\gamma K} \).

CTR II can perform the synthesis of the transfer rules for constants under arbitrary user-defined side conditions automatically. However, the algorithm guarantees neither that it can identify whether a transfer rule exists for a given constant under a given set of side conditions, nor that it will be found if it does exist.
C Acronyms

The most important acronyms that are used in this article are summarised in Table 3.

References

München, Munich, Germany.


