Quotients of Bounded Natural Functors

Basil Fürer[1], Andreas Lochbihler[2], Joshua Schneider[1], and Dmitriy Traytel[1]

[1] Institute of Information Security, Department of Computer Science, ETH Zürich, Switzerland
[2] Digital Asset (Switzerland) GmbH, Zurich, Switzerland

Abstract. The functorial structure of type constructors is the foundation for many definition and proof principles in higher-order logic (HOL). For example, inductive and coinductive datatypes can be built modularly from bounded natural functors (BNFs), a class of well-behaved type constructors. Composition, fixpoints, and—under certain conditions—subtypes are known to preserve the BNF structure. In this paper, we tackle the preservation question for quotients, the last important principle for introducing new types in HOL. We identify sufficient conditions under which a quotient inherits the BNF structure from its underlying type. We extend the Isabelle proof assistant with a command that automates the registration of a quotient type as a BNF by lifting the underlying type's BNF structure. We demonstrate the command's usefulness through several case studies.

1 Introduction

The functorial structure of type constructors forms the basis for many definition and proof principles in proof assistants. Examples include datatype and codatatype definitions [3, 9, 37], program synthesis [13, 19, 24], generalized term rewriting [36], and reasoning based on representation independence [6, 19, 23] and about effects [26, 27].

A type constructor becomes a functor through a mapper operation that lifts functions on the type arguments to the constructed type. The mapper must be *functorial*, i.e., preserve identity functions (id) and distribute over function composition ($\circ$). For example, the list type constructor $\_\ list$[1] has the well-known mapper $\mathsf{map} :: (\alpha \to \beta) \to \alpha\ list \to \beta\ list$, which applies the given function to every element in the given list. It is functorial:

$$\mathsf{map}\ \mathsf{id} = \mathsf{id} \qquad\qquad \mathsf{map}\ g \circ \mathsf{map}\ f = \mathsf{map}\ (g \circ f)$$

Most applications of functors can benefit from even richer structures. In this paper, we focus on bounded natural functors (BNFs) [37] (Section 2.1). A BNF comes with additional setter operators that return sets of occurring elements, called atoms, for each type argument. The setters must be *natural* transformations, i.e., commute with the mapper, and *bounded*, i.e., have a fixed cardinality bound on the sets they return. For example, $\mathsf{set} :: \alpha\ list \to \alpha\ set$ returns the set of elements in a list. It satisfies $\mathsf{set} \circ \mathsf{map}\ f = f\langle\_\rangle \circ \mathsf{set}$, where $f\langle\_\rangle$ denotes the function that maps a set X to $f\langle X\rangle = \{f\ x \mid x \in X\}$, i.e., the image of X under f. Moreover, since lists are finite sequences, $\mathsf{set}\ xs$ is always a finite set.

Originally, BNFs were introduced for modularly constructing datatypes and codatatypes [9] in the Isabelle/HOL proof assistant. Although (co)datatypes are still the most important use case, the BNF structure is used nowadays in other contexts such as reasoning via free theorems [29] and transferring theorems between types [22, 28].

[1] Type constructors are written postfix in this paper.

Several type definition principles in HOL preserve the BNF structure: composition (e.g., $(\alpha\ \mathit{list})\ \mathit{list}$), datatypes and codatatypes [37], and—under certain conditions—subtypes [7,28]. Subtypes include records and type copies. Accordingly, when a new type constructor is defined via one of these principles from an existing BNF, then the new type automatically comes with a mapper and setters and with theorems for the BNF properties.

One important type definition principle is missing above: quotients [18,19,21,34,35] (Section 2.2). A quotient type identifies elements of an underlying type according to a (partial) equivalence relation $\sim$. That is, the quotient type is isomorphic to the equivalence classes of $\sim$. For example, unordered pairs $\alpha\ \mathit{upair}$ are the quotient of ordered pairs $\alpha \times \alpha$ and the equivalence relation $\sim_{\mathit{upair}}$ generated by $(x,y) \sim_{\mathit{upair}} (y,x)$. Similarly, finite sets, bags, and cyclic lists are quotients of lists where the equivalence relation permutes or duplicates the list elements as needed.

In this paper, we answer the question when and how a quotient type inherits its underlying type's BNF structure. It is well known that a quotient preserves the functorial properties if the underlying type's mapper preserves $\sim$; then the quotient type's mapper is simply the lifting of the underlying type's mapper to equivalence classes [3]. For setters, the situation is more complicated. We discovered that if the setters are defined as one would expect, the resulting structure may not preserve empty intersections, i.e., it is *unsound* in Adámek et al.'s [2] terminology. All BNFs, however, are sound. To repair the situation, we characterize the setters in terms of the mapper and identify a definition scheme for the setters that results in sound functors. We then derive sufficient conditions on the equivalence relation $\sim$ for the BNF properties to be preserved for these definitions (Section 3). With few exceptions, we omit proofs and refer to our technical report [15], which contains them.

Moreover, we have implemented an Isabelle/HOL command that automates the registration of a quotient type as a BNF (Section 4); the user merely needs to discharge the conditions on $\sim$. One of the conditions, subdistributivity, often requires considerable proof effort, though. We therefore developed a novel sufficient criterion using confluent relations that simplifies the proofs in our case studies (Section 3.4). Our implementation is distributed with the Isabelle2020 release.

Contributions The main contributions of this paper are the following:

1. We identify sufficient criteria for when a quotient type preserves the BNF properties of the underlying type. Registering a quotient as a BNFs allows (co)datatypes to nest recursion through it. Consider for example node-labeled unordered binary trees

   $$\texttt{datatype}\ \mathit{ubtree} = \mathsf{Leaf} \mid \mathsf{Node}\ \mathit{nat}\ (\mathit{ubtree}\ \mathit{upair})$$

   BNF use cases beyond datatypes benefit equally.
2. In particular, we show that the straightforward definitions would cause the functor to be unsound, and find better definitions that avoid unsoundness. This problem is not limited to BNFs. The lifting operations for Lean's QPFs [3] also suffer from unsoundness and our repair applies to them as well (Section 5.2).
3. We propose a sufficient criterion on $\sim$ for subdistributivity, which is typically the most difficult BNF property to show. We show with several examples that the criterion is applicable in practice and yields relatively simple proofs.

4. We have implemented an Isabelle/HOL command to register the quotient as a BNF, once the user has discharged the conditions on $\sim$. The command also generates proof rules for transferring theorems about the BNF operations from the underlying type to the quotient (Section 4.2). Several case studies demonstrate the command's usefulness. Some examples reformulate well-known BNFs as quotients (e.g., unordered pairs, distinct lists, finite sets). Others formally prove the BNF properties for the first time, e.g., cyclic lists, the free idempotent monoid, and regular expressions modulo ACI. These examples become part of the collection of formalized BNFs and can thus be used in datatype definitions and other BNF applications.

Example 1. To illustrate our contribution's usefulness, we consider linear dynamic logic (LDL) [14], an extension of linear temporal logic with regular expressions. LDL's syntax is usually given as two mutually recursive datatypes of formulas and regular expressions [5,14]. Here, we opt for nested recursion, which has the modularity benefit of being able to formalize regular expressions separately. We define regular expressions $\alpha\ re$:

$$\texttt{datatype}\ \alpha\ re = \mathsf{Atom}\ \alpha \mid \mathsf{Alt}\ (\alpha\ re)\ (\alpha\ re) \mid \mathsf{Conc}\ (\alpha\ re)\ (\alpha\ re) \mid \mathsf{Star}\ (\alpha\ re)$$

Often, it is useful to consider regular expressions modulo some syntactic equivalences. For example, identifying expressions modulo the associativity, commutativity, and idempotence (ACI) of the alternation constructor Alt results in a straightforward construction of deterministic finite automata from regular expressions via Brzozowski derivatives [32]. We define the ACI-equivalence $\sim_{aci}$ as the least congruence relation satisfying:

$$\mathsf{Alt}\ (\mathsf{Alt}\ r\ s)\ t \sim_{aci} \mathsf{Alt}\ r\ (\mathsf{Alt}\ s\ t) \qquad \mathsf{Alt}\ r\ s \sim_{aci} \mathsf{Alt}\ s\ r \qquad \mathsf{Alt}\ r\ r \sim_{aci} r$$

Next, we define the quotient type of regular expressions modulo ACI $\alpha\ re_{aci}$ and the datatype of LDL formulas ldl, which uses nested recursion through $\alpha\ re_{aci}$.

$$\texttt{quotient\_type}\ \alpha\ re_{aci} = \alpha\ re/{\sim_{aci}}$$
$$\texttt{datatype}\ ldl = \mathsf{Prop}\ string \mid \mathsf{Neg}\ ldl \mid \mathsf{Conj}\ ldl\ ldl \mid \mathsf{Match}\ (ldl\ re_{aci})$$

For the last declaration to succeed, Isabelle must know that $\alpha\ re_{aci}$ is a BNF. We will show in Section 3.4 how our work allows us to lift $\alpha\ re$'s BNF structure to $\alpha\ re_{aci}$. $\diamond$

2 Background

We work in Isabelle/HOL, Isabelle's variant of classical higher-order logic—a simply typed theory with Hilbert choice and rank-1 polymorphism. We refer to a textbook for a detailed introduction to Isabelle/HOL [31] and only summarize relevant notation here.

Types are built from type variables $\alpha, \beta, \ldots$ via type constructors. A type constructor can be nullary (nat) or have some type arguments ($\alpha\ list$, $\alpha\ set$, $(\alpha,\beta)\ upair$). Type constructor application is written postfix. Exceptions are the binary type constructors for sums ($+$), products ($\times$), and functions ($\to$), all written infix. Terms are built from variables $x, y, \ldots$ and constants $\mathsf{c}, \mathsf{d}, \ldots$ via lambda-abstractions $\lambda x.\ t$ and applications $t\ u$. The sum type's embeddings are Inl and Inr and the product type's projections are fst and snd.

The primitive way of introducing new types in HOL is to take a non-empty subset of an existing type. For example, the type of lists could be defined as the set of pairs $(n :: nat,\ f :: nat \to \alpha)$ where n is the list's length and $f\ i$ is the list's ith element for $i < n$ and some fixed unspecified element of type α for $i \geq n$. To spare the users from such low-level encodings, Isabelle/HOL offers more high-level mechanisms for introducing new types, which are internally reduced to primitive subtyping. In fact, lists are defined as an inductive `datatype` $\alpha\ list = [] \mid \alpha \# \alpha\ list$, where $[]$ is the empty list and # is the infix list constructor. Recursion in datatypes and their coinductive counterparts may take place only under well-behaved type constructors, the bounded natural functors (Section 2.1). Quotient types (Section 2.2) are another high-level mechanism for introducing new types.

For n-ary definitions, we use the vector notation $\overline{x}$ that denotes $x_1, \ldots, x_n$ where n is clear from the context. Vectors spanning several variables indicate repetition with synchronized indices. For example, $\mathsf{map}_F\ \overline{(g \circ f)}$ abbreviates $\mathsf{map}_F\ (g_1 \circ f_1)\ \ldots\ (g_n \circ f_n)$. Abusing notation slightly, we write $\overline{\alpha} \to \beta$ for the n-ary function type $\alpha_1 \to \cdots \to \alpha_n \to \beta$.

To simplify notation, we identify the type of binary predicates $\alpha \to \beta \to \mathbb{B}$ and sets of pairs $(\alpha \times \beta)\ set$, and write $\alpha \otimes \beta$ for both. These types are different in Isabelle/HOL and the BNF ecosystem works with binary predicates. The identification allows us to use set operations, e.g., the subset relation $\subseteq$ or relation composition $\bullet$ (both written infix).

2.1 Bounded Natural Functors

A bounded natural functor (BNF) [37] is an n-ary type constructor $\overline{\alpha}\ F$ equipped with the following polymorphic constants. Here and elsewhere, i implicitly ranges over $\{1, \ldots, n\}$:

$$\begin{array}{ll} \mathsf{map}_F :: \overline{(\alpha \to \beta)} \to \overline{\alpha}\ F \to \overline{\beta}\ F & \quad \mathsf{bd}_F :: bd\_type \otimes bd\_type \\ \mathsf{set}_{F,i} :: \overline{\alpha}\ F \to \alpha_i\ set \quad \text{for all } i & \quad \mathsf{rel}_F :: \overline{(\alpha \otimes \beta)} \to \overline{\alpha}\ F \otimes \overline{\beta}\ F \end{array}$$

The *shape and content* intuition [37] is a useful way of thinking about elements of $\overline{\alpha}\ F$. The mapper map_F leaves the shape unchanged but modifies the contents by applying its function arguments. The n setters $\mathsf{set}_{F,i}$ extract the contents (and dispose of the shape). For example, the shape of a list is given by its length, which map preserves. The cardinal bound bd_F is a fixed bound on the number of elements returned by $\mathsf{set}_{F,i}$. Cardinal numbers are represented in HOL using particular well-ordered relations [10]. Finally, the relator rel_F lifts relations on the type arguments to a relation on $\overline{\alpha}\ F$ and $\overline{\beta}\ F$. Thereby, it only relates elements of $\overline{\alpha}\ F$ and $\overline{\beta}\ F$ that have the same shape.

The BNF constants must satisfy the following properties:

$$\begin{array}{rl} \text{MAP\_ID} & \mathsf{map}_F\ \overline{\mathsf{id}} = \mathsf{id} \\ \text{MAP\_COMP} & \mathsf{map}_F\ \overline{g} \circ \mathsf{map}_F\ \overline{f} = \mathsf{map}_F\ \overline{(g \circ f)} \\ \text{SET\_MAP} & \mathsf{set}_{F,i} \circ \mathsf{map}_F\ \overline{f} = f_i \langle \_ \rangle \circ \mathsf{set}_{F,i} \\ \text{MAP\_CONG} & (\forall i.\ \forall z \in \mathsf{set}_{F,i}\ x.\ f_i\ z = g_i\ z) \implies \mathsf{map}_F\ \overline{f}\ x = \mathsf{map}_F\ \overline{g}\ x \\ \text{SET\_BD} & |\mathsf{set}_{F,i}\ x| \leq_o \mathsf{bd}_F \\ \text{BD} & \mathsf{infinite\_card}\ \mathsf{bd}_F \\ \text{IN\_REL} & \mathsf{rel}_F\ \overline{R}\ x\ y = \exists z.\ (\forall i.\ \mathsf{set}_{F,i}\ z \subseteq R_i) \land \mathsf{map}\ \overline{\mathsf{fst}}\ z = x \land \mathsf{map}\ \overline{\mathsf{snd}}\ z = y \\ \text{REL\_COMP} & \mathsf{rel}_F\ \overline{R} \bullet \mathsf{rel}_F\ \overline{S} \subseteq \mathsf{rel}_F\ \overline{(R \bullet S)} \end{array}$$

Properties MAP_ID and MAP_COMP capture the mapper's functoriality; SET_MAP the setters' naturality. Moreover, the mapper and the setters must agree on what they identify as content (MAP_CONG). Any set returned by $\mathsf{set}_{F,i}$ must be bounded (SET_BD); the operator $\leq_o$ compares cardinal numbers [10]. The bound is required to be infinite (BD), which simplifies arithmetics. The relator can be expressed in terms of the mapper and the setter (IN_REL) and must distribute over relation composition (REL_COMP). The other inclusion, namely $\mathsf{rel}_F\ \overline{(R \bullet S)} \subseteq \mathsf{rel}_F\ \overline{R} \bullet \mathsf{rel}_F\ \overline{S}$, follows from these properties. We refer to REL_COMP as *subdistributivity* because it only requires one inclusion.

A useful derived operator is the action on sets $\boxed{F} :: \overline{\alpha\ set} \to \overline{\alpha}\ F\ set$, which generalizes the type constructor's action on its type arguments. Formally, $\boxed{F}\,\overline{A} = \{x \mid \forall i.\ \mathsf{set}_{F,i}\ x \subseteq A_i\}$. Note that we can write $z \in \boxed{F}\,\overline{R}$ to replace the equivalent $\forall i.\ \mathsf{set}_{F,i}\ z \subseteq R_i$ in IN_REL.

Most basic types are BNFs, notably, sum and product types. BNFs are closed under composition, e.g., $1+\alpha\times\beta$ is a BNF with the mapper $\lambda f\ g.\ \mathsf{map}_{1+}\ (\mathsf{map}_{\times}\ f\ g)$, where 1 is the unit type (consisting of the single element $\star$) and $\mathsf{map}_{1+}\ h = \mathsf{map}_{+}\ \mathsf{id}\ h$. Moreover, BNFs support fixpoint operations, which correspond to (co)datatypes, and are closed under them [37]. For instance, the `datatype` command internally computes a least solution for the fixpoint type equation $\beta = 1+\alpha\times\beta$ to define the $\alpha\ list$ type. Closure means that the resulting datatype, here $\alpha\ list$, is equipped with the BNF structure, e.g., the mapper map. Also subtypes inherit the BNF structure under certain conditions [7]. For example, the subtype $\alpha\ nelist$ of non-empty lists $\{xs :: \alpha\ list \mid xs \neq []\}$ is a BNF.

2.2 Quotient types

An equivalence relation $\sim$ on a type T partitions the type into equivalence classes. Isabelle/HOL supports the definition of the quotient type $Q = T/{\sim}$, which yields a new type Q isomorphic to the set of equivalence classes [21]. For example, consider $\sim_{fset}$ that relates two lists if they have the same set of elements, i.e., $xs \sim_{fset} ys$ iff $\mathsf{set}\ xs = \mathsf{set}\ ys$. The following command defines the type $\alpha\ fset$ of finite sets as a quotient of lists:

`quotient_type` $\alpha\ fset = \alpha\ list/{\sim_{fset}}$

This command requires a proof that $\sim_{fset}$ is, in fact, an equivalence relation.

The Lifting and Transfer tools [19,22] automate the lifting of definitions and theorems from the raw type T to the quotient Q. For example, the image operation on finite sets can be obtained by lifting the list mapper map using the command

`lift_definition` $\mathsf{fimage} :: (\alpha \to \beta) \to \alpha\ fset \to \beta\ fset$ `is map`

Lifting is only possible for terms that respect the quotient. For fimage, respectfulness states that $\mathsf{map}\ f\ xs \sim_{fset} \mathsf{map}\ f\ ys$ whenever $xs \sim_{fset} ys$.

Lifting and Transfer are based on *transfer rules* that relate two terms of possibly different types. The `lift_definition` command automatically proves the transfer rule

$$(\mathsf{map}, \mathsf{fimage}) \in ((=) \Rrightarrow \mathsf{cr}_{fset} \Rrightarrow \mathsf{cr}_{fset})$$

where $A \Rrightarrow B$ (right-associative) relates two functions iff they map A-related arguments to B-related results. The correspondence relation cr_{fset} relates a list with the finite set that it represents, i.e., the set whose corresponding equivalence class contains the list. Every quotient is equipped with such a correspondence relation. The meaning of the above rule is that applying $\mathsf{map}\ f$ to a list representing the finite set X results in a list that represents $\mathsf{fimage}\ f\ X$, for all f. The transfer rule's relation $(=) \Rrightarrow \mathsf{cr}_{fset} \Rrightarrow \mathsf{cr}_{fset}$ is constructed according to the types of the related terms. This enables the composition of transfer rules to relate larger terms. For instance, the Transfer tool automatically derives

$$(\forall x.\ \mathsf{set}\ (\mathsf{map}\ \mathsf{id}\ x) = \mathsf{set}\ x) \longleftrightarrow (\forall X.\ \mathsf{fimage}\ \mathsf{id}\ X = X)$$

such that the equation $\forall X.\ \mathsf{fimage}\ \mathsf{id}\ X = X$ can be proved by reasoning about lists.

3 Quotients of Bounded Natural Functors

We develop the theory for when a quotient type inherits the underlying type's BNF structure. We consider the quotient $\overline{\alpha}\ Q = \overline{\alpha}\ F/{\sim}$ of an n-ary BNF $\overline{\alpha}\ F$ over an equivalence relation $\sim$ on $\overline{\alpha}\ F$. The first idea is to define map_Q and $\mathsf{set}_{Q,i}$ in terms of F's operations:

```
quotient_type $\overline{\alpha}\ Q = \overline{\alpha}\ F/{\sim}$
lift_definition $\mathsf{map}_Q :: \overline{(\alpha \to \beta)} \to \overline{\alpha}\ Q \to \overline{\beta}\ Q$ is $\mathsf{map}_F$
lift_definition $\mathsf{set}_{Q,i} :: \overline{\alpha}\ Q \to \alpha_i\ set$ is $\mathsf{set}_{F,i}$
```

These three commands require the user to discharge the following proof obligations:

$$\mathsf{equivp} \sim \qquad (1) \qquad\qquad x \sim y \Longrightarrow \mathsf{map}_F\ \overline{f}\ x \sim \mathsf{map}_F\ \overline{f}\ y \qquad (2)$$

$$x \sim y \Longrightarrow \mathsf{set}_{F,i}\ x = \mathsf{set}_{F,i}\ y \qquad (3)$$

The first two conditions are as expected: $\sim$ must be an equivalence relation, by (1), and compatible with F's mapper, by (2), i.e., map_F preserves $\sim$. The third condition, however, demands that equivalent values contain the same atoms. This rules out many practical examples including the following simplified (and therefore slightly artificial) one.

Example 2. Consider $\alpha\ F_P = \alpha + \alpha$ with the equivalence relation $\sim_P$ generated by $\mathsf{Inl}\ x \sim_P \mathsf{Inl}\ y$, where Inl is the sum type's left embedding. That is, $\sim_P$ identifies all values of the form $\mathsf{Inl}\ z$ and thus $\alpha\ Q_P = \alpha\ F_P/{\sim_P}$ is isomorphic to the type $1 + \alpha$. However, $\mathsf{Inl}\ x$ and $\mathsf{Inl}\ y$ have different sets of atoms $\{x\}$ and $\{y\}$, assuming $x \neq y$. ◇

We derive better definitions for the setters and conditions under which they preserve the BNF properties. To that end, we characterize setters in terms of the mapper (Section 3.1). Using this characterization, we derive the relationship between $\mathsf{set}_{Q,i}$ and $\mathsf{set}_{F,i}$ and identify the conditions on $\sim$ (Section 3.2). Next, we do the same for the relator (Section 3.3). We thus obtain the conditions under which $\overline{\alpha}\ Q$ preserves F's BNF properties. One of the conditions, the relator's subdistributivity over relation composition, is often difficult to show directly in practice. We therefore present an easier-to-establish criterion for the special case where a confluent rewrite relation $\rightsquigarrow$ over-approximates $\sim$ (Section 3.4).

3.1 Characterization of the BNF setter

We now characterize $\mathsf{set}_{F,i}$ in terms of map_F for an arbitrary BNF $\overline{\alpha}\ F$. Observe that F's action $\boxed{F}\,\overline{A}$ on sets contains all values that can be built with atoms from $\overline{A}$. Hence, $\mathsf{set}_{F,i}\ x$ is the smallest set A_i such that x can be built from atoms in A_i. Formally:

$$\mathsf{set}_{F,i}\ x = \bigcap \{A_i \mid x \in \boxed{F}\ \overline{\mathsf{UNIV}}\ A_i\ \overline{\mathsf{UNIV}}\} \qquad (4)$$

Only atoms of type α_i are restricted; all other atoms α_j may come from UNIV, the set of all elements of type α_j. Moreover, $\boxed{F}$ can be defined without $\mathsf{set}_{F,i}$, namely by trying to distinguish values using the mapper. Informally, x contains atoms not from $\overline{A}$ iff $\mathsf{map}_F\ \overline{f}\ x$ differs from $\mathsf{map}_F\ \overline{g}\ x$ for some functions $\overline{f}$ and $\overline{g}$ that agree on $\overline{A}$. Hence, we obtain:

$$\boxed{F}\,\overline{A} = \{x \mid \forall \overline{f}\ \overline{g}.\ (\forall i.\ \forall a \in A_i.\ f_i\ a = g_i\ a) \longrightarrow \mathsf{map}_F\ \overline{f}\ x = \mathsf{map}_F\ \overline{g}\ x\} \qquad (5)$$

Proof. From left to right is trivial with MAP_CONG. So let x be such that $\mathsf{map}_F\ \overline{f}\ x = \mathsf{map}_F\ \overline{g}\ x$ whenever $f_i\ a = g_i\ a$ for all $a \in A_i$ and all i. By the definition of $\boxed{F}$, it suffices to show that $\mathsf{set}_{F,i}\ x \subseteq A_i$. Set $f_i\ a = (a \in A_i)$ and $g_i\ a = \mathsf{True}$. Then,

$$\begin{aligned}
f_i\langle \mathsf{set}_{F,i}\ x\rangle &= \mathsf{set}_{F,i}\ (\mathsf{map}_F\ \overline{f}\ x) && \text{by SET\_MAP}\\
&= \mathsf{set}_{F,i}\ (\mathsf{map}_F\ \overline{g}\ x) && \text{by choice of } x \text{ as } \overline{f} \text{ and } \overline{g} \text{ agree on } \overline{A}\\
&= (\lambda\_.\,\mathsf{True})\langle \mathsf{set}_{F,i}\ x\rangle && \text{by SET\_MAP}\\
&\subseteq \{\mathsf{True}\}
\end{aligned}$$

Therefore, $\forall a \in \mathsf{set}_{F,i}\ x.\ f_i\ a$, i.e., $\mathsf{set}_{F,i}\ x \subseteq A_i$. □

Equations 4 and 5 reduce the setters $\mathsf{set}_{F,i}$ of a BNF to its mapper map_F. In the next section, we will use this characterization to derive a definition of $\mathsf{set}_{Q,i}$ in terms of $\mathsf{set}_{F,i}$. Yet, this definition does not give us naturality out of the box.

Example 3 ([2, Example 4.2, part iii]). Consider the functor $\alpha\ F_{ae} = nat \to \alpha$ of infinite sequences with $x \sim_{ae} y$ whenever $\{n \mid x\ n \neq y\ n\}$ is finite. That is, two sequences are equivalent iff they are equal almost everywhere. Conditions (1) and (2) hold, but not the naturality for the corresponding map_Q and set_Q. ◇

Gumm [16] showed that set_F as defined in terms of (4) and (5) is a natural transformation iff $\boxed{F}$ preserves wide intersections and preimages, i.e.,

$$\boxed{F}\ \overline{(\bigcap \mathscr{A})} = \bigcap\{\boxed{F}\ \overline{A} \mid \forall i.\ A_i \in \mathscr{A}_i\} \tag{6}$$

$$\boxed{F}\ \overline{(f^{-1}\langle A\rangle)} = (\mathsf{map}_F\ \overline{f})^{-1}\langle \boxed{F}\ \overline{A}\rangle \tag{7}$$

where $f^{-1}\langle A\rangle = \{x \mid f\ x \in A\}$ denotes the preimage of A under f. Then, $\boxed{F}\ \overline{A} = \{x \mid \forall i.\ \mathsf{set}_{F,i}\ x \subseteq A_i\}$ holds. The quotient in Example 3 does not preserve wide intersections.

In theory, we have now everything we need to define the BNF operations on the quotient $\overline{\alpha}\ Q = \overline{\alpha}\ F/{\sim}$: Define map_Q as the lifting of map_F. Define $\boxed{Q}$ and $\mathsf{set}_{Q,i}$ using (5) and (4) in terms of map_Q, and the relator via IN_REL. Prove that $\boxed{Q}$ preserves preimages and wide intersections. Prove that rel_Q satisfies subdistributivity (REL_COMP).

Unfortunately, the definitions and the preservation conditions are phrased in terms of Q, not in terms of F and $\sim$. It is therefore unclear how $\mathsf{set}_{Q,i}$ and rel_Q relate to $\mathsf{set}_{F,i}$ and rel_F. In practice, understanding this relationship is important: we want to express the BNF operations and discharge the proof obligations in terms of F's operations and later use the connection to transfer properties from set_F and rel_F to set_Q and rel_Q. We will work out the precise relationships for the setters in Section 3.2 and for the relator in Section 3.3.

3.2 The quotient's setter

We relate Q's setters to F's operations and $\sim$. We first look at $\boxed{Q}$, which characterizes $\mathsf{set}_{Q,i}$ via (4). Let $[x]_\sim = \{y \mid x \sim y\}$ denote the equivalence class that $x :: \overline{\alpha}\ F$ belongs to, and $[A]_\sim = \{[x]_\sim \mid x \in A\}$ denote the equivalence classes of elements in A. We identify the values of $\overline{\alpha}\ Q$ with $\overline{\alpha}\ F$'s equivalence classes. Then, it follows using (1), (2), and (5) that $\boxed{Q}\, A = [\colorbox{gray}{F}\, A]_\sim$ where

$$\colorbox{gray}{F}\ \overline{A} = \{x \mid \forall \overline{f}\ \overline{g}.\ (\forall i.\ \forall a \in A_i.\ f_i\ a = g_i\ a) \longrightarrow \mathsf{map}_F\ \overline{f}\ x \sim \mathsf{map}_F\ \overline{g}\ x\} \tag{8}$$

Equation 8 differs from (5) only in that the equality in $\mathsf{map}_F\ \overline{f}\ x = \mathsf{map}_F\ \overline{g}\ x$ is replaced by $\sim$. Clearly $[\boxed{F}\,\overline{A}]_\sim \subseteq [\colorbox{gray}{F}\,\overline{A}]_\sim$. The converse does not hold in general, as shown next.

Example 2 (continued). For the example viewing $1+\alpha$ as a quotient of $\alpha\ F_P = \alpha + \alpha$ via $\sim_P$, we have $[\mathsf{Inl}\ x]_\sim \in [\colorbox{gray}{F_P}\,\{\}]_{\sim_P}$ because $\mathsf{map}_{F_P}\ f\ (\mathsf{Inl}\ x) = \mathsf{Inl}\ (f\ x) \sim_P \mathsf{Inl}\ (g\ x) = \mathsf{map}_{F_P}\ g\ (\mathsf{Inl}\ x)$ for all f and g. Yet, $\boxed{F_P}\,\{\}$ is empty, and so is $[\boxed{F_P}\,\{\}]_{\sim_P}$. $\diamond$

This problematic behavior occurs only for empty sets A_i. To avoid it, we change types: Instead of $\overline{\alpha}\ F/{\sim}$, we consider the quotient $\overline{(1+\alpha)}\ F/{\sim}$, where $1+\alpha_i$ adds a new atom $\textcircled{$\star$} = \mathsf{Inl}\ \star$ to the atoms of type α_i. We write $\mathfrak{e} :: \alpha \to 1+\alpha$ for the embedding of α into $1+\alpha$ (i.e., $\mathfrak{e} = \mathsf{Inr}$). Then, we have the following equivalence:

Lemma 1. $\colorbox{gray}{$F$}\,\overline{A} = \{x \mid [\mathsf{map}_F\ \overline{\mathfrak{e}}\ x]_\sim \in [\boxed{F}\,\overline{(\{\textcircled{$\star$}\} \cup \mathfrak{e}\langle A\rangle)}]_\sim\}$.

Proof. From left to right: Let $x \in \colorbox{gray}{$F$}\,\overline{A}$ and set $f_i\ y = \mathfrak{e}\ y$ for $y \in A_i$ and $f_i\ y = \textcircled{$\star$}$ for $y \notin A_i$. Then, $\mathsf{set}_{F,i}\ (\mathsf{map}_F\ \overline{f}\ x) = f_i\langle \mathsf{set}_{F,i}\ x\rangle$ by the naturality of $\mathsf{set}_{F,i}$ and $f_i\langle B\rangle \subseteq \{\textcircled{$\star$}\} \cup \mathfrak{e}\langle A_i\rangle$ by f_i's definition for any B. Hence $\mathsf{map}\ \overline{f}\ x \in \boxed{F}\,\overline{(\{\textcircled{$\star$}\} \cup \mathfrak{e}\langle A\rangle)}$ as $\boxed{F}\,\overline{A} = \{x \mid \forall i.\ \mathsf{set}_{F,i}\ x \subseteq A_i\}$ by the BNF properties. So, $[\mathsf{map}_F\ \overline{\mathfrak{e}}\ x]_\sim \in [\boxed{F}\,\overline{(\{\textcircled{$\star$}\} \cup \mathfrak{e}\langle A\rangle)}]_\sim$ because $\mathsf{map}_F\ \overline{\mathfrak{e}}\ x \sim \mathsf{map}\ \overline{f}\ x$ by (8) and $x \in \colorbox{gray}{$F$}\,\overline{A}$.

From right to left: Let x such that $\mathsf{map}_F\ \overline{\mathfrak{e}}\ x \sim y$ for some $y \in \boxed{F}\,\overline{(\{\textcircled{$\star$}\} \cup \mathfrak{e}\langle A\rangle)}$. Let $\overline{f}$ and $\overline{g}$ such that $f_i\ a = g_i\ a$ for all $a \in A_i$ and all i. Then, $\mathsf{map}_F\ \overline{f}\ x \sim \mathsf{map}_F\ \overline{g}\ x$ holds by the following reasoning, where $\mathfrak{e}^{-1}$ denotes the left-inverse of $\mathfrak{e}$ and $\mathsf{map}_{1+}\ h$ satisfies $\mathsf{map}_{1+}\ h\ (\mathfrak{e}\ a) = \mathfrak{e}\ (h\ a)$ and $\mathsf{map}_{1+}\ h\ \textcircled{$\star$} = \textcircled{$\star$}$:

$$\begin{aligned}
\mathsf{map}_F\ \overline{f}\ x &= \mathsf{map}_F\ \overline{\mathfrak{e}^{-1}}\ (\mathsf{map}_F\ \overline{(\mathsf{map}_{1+}\ f)}\ (\mathsf{map}_F\ \overline{\mathfrak{e}}\ x)) && \text{as } f_i = \mathfrak{e}^{-1} \circ \mathsf{map}_{1+}\ f_i \circ \mathfrak{e}\\
&\sim \mathsf{map}_F\ \overline{\mathfrak{e}^{-1}}\ (\mathsf{map}_F\ \overline{(\mathsf{map}_{1+}\ f)}\ y) && \text{by } \mathsf{map}_F\ \overline{\mathfrak{e}}\ x \sim y \text{ and (2)}\\
&= \mathsf{map}_F\ \overline{\mathfrak{e}^{-1}}\ (\mathsf{map}_F\ \overline{(\mathsf{map}_{1+}\ g)}\ y) && \text{by choice of } y \text{ and (5)}\\
&\sim \mathsf{map}_F\ \overline{\mathfrak{e}^{-1}}\ (\mathsf{map}_F\ \overline{(\mathsf{map}_{1+}\ g)}\ (\mathsf{map}_F\ \overline{\mathfrak{e}}\ x)) && \text{by } y \sim \mathsf{map}_F\ \overline{\mathfrak{e}}\ x \text{ and (2)}\\
&= \mathsf{map}_F\ \overline{g}\ x && \text{as } \mathfrak{e}^{-1} \circ \mathsf{map}_{1+}\ g_i \circ \mathfrak{e} = g_i \quad \square
\end{aligned}$$

Lemma 1 allows us to express the conditions (6) and (7) on $\boxed{Q}$ in terms of $\sim$ and $\boxed{F}$. For wide intersections, the condition is as follows (the other inclusion holds trivially):

$$\forall i.\ \mathscr{A}_i \neq \{\} \wedge \left(\bigcap \mathscr{A}_i \neq \{\}\right) \Longrightarrow \bigcap\{[\boxed{F}\,\overline{A}]_\sim \mid \forall i.\ A_i \in \mathscr{A}_i\} \subseteq \left[\bigcap\{\boxed{F}\,\overline{A} \mid \forall i.\ A_i \in \mathscr{A}_i\}\right]_\sim \quad (9)$$

The conclusion is as expected: for sets of the form $\boxed{F}\,\overline{A}$, taking equivalence classes preserves wide intersections. The assumption is the interesting part: preservation is needed only for *non-empty* intersections. Non-emptiness suffices because Lemma 1 relates $\colorbox{gray}{$F$}\,\overline{A}$ to $\boxed{F}\,\overline{(\{\textcircled{$\star$}\} \cup \mathfrak{e}\langle A\rangle)}$ and all intersections of interest therefore contain $\textcircled{$\star$}$. (The condition does not explicitly mention $\textcircled{$\star$}$ because Lemma 1 holds for any element that is not in A.)

Condition 9 is satisfied trivially for equivalence relations that preserve $\mathsf{set}_{F,i}$, i.e., satisfy (3). Examples include permutative structures like finite sets and cyclic lists.

Lemma 2. *Suppose that* $\sim$ *satisfies* (3)*. Then,* $[\boxed{F}\,\overline{A}]_\sim = \boxed{F}\,\overline{A}$ *and condition* (9) *holds.*

In contrast, the non-emptiness assumption is crucial for quotients that identify values with different sets of atoms, such as Example 2. In general, such quotients do *not* preserve empty intersections (Section 5).

We can factor condition (9) into a separate property for each type argument i:

$$\mathscr{A}_i \neq \{\} \wedge (\bigcap \mathscr{A}_i \neq \{\}) \Longrightarrow \bigcap_{A \in \mathscr{A}_i} [\{x \mid \mathsf{set}_{F,i}\ x \subseteq A\}]_\sim \subseteq \left[\{x \mid \mathsf{set}_{F,i}\ x \subseteq \bigcap \mathscr{A}_i\}\right]_\sim \quad (10)$$

This form is used in our implementation (Section 4). It is arguably more natural to prove for a concrete functor F because each property focuses on a single setter.

Lemma 3. *Suppose that* $\sim$ *satisfies* (1) *and* (2)*. Then,* (9) *holds iff* (10) *holds for all i.*

Preservation of preimages amounts to the following unsurprising condition:

$$\forall i.\ f_i^{-1}\langle A_i\rangle \neq \{\} \implies (\mathsf{map}_F\ \overline{f})^{-1}\left\langle \bigcup [\boxed{F}\,\overline{A}]_\sim \right\rangle \subseteq \bigcup \left[(\mathsf{map}_F\ \overline{f})^{-1}\langle \boxed{F}\,\overline{A}\rangle\right]_\sim \quad (11)$$

As for wide intersections, taking equivalence classes must preserve *non-empty* preimages (the inclusion from right to left holds trivially). Again, non-emptiness comes from ⍟ being contained in all sets of interest. We do not elaborate on preimage preservation any further as it follows from subdistributivity, which we will look at in the next subsection.

Under conditions (9) and (11), we obtain the following characterization for set_Q:

Theorem 1 (Setter characterization). $\mathit{set}_{Q,i}\ [x]_\sim = \bigcap_{y \in [\mathit{map}_F\ \mathfrak{e}\ x]_\sim} \{a \mid \mathfrak{e}\ a \in \mathit{set}_{F,i}\ y\}$

3.3 The quotient's relator

In the previous section, we have shown that it is not a good idea to naively lift the setter and a more general construction is needed. We now show that the same holds for the relator. The following straightforward definition

$\texttt{lift\_definition}\ \mathsf{rel}_Q :: \overline{(\alpha \otimes \beta)} \to \overline{\alpha}\ Q \otimes \overline{\beta}\ Q\ \texttt{is}\ \mathsf{rel}_F$

relates two equivalence classes $[x]_\sim$ and $[y]_\sim$ iff there are representatives $x' \in [x]_\sim$ and $y' \sim [y]_\sim$ such that $(x', y') \in \mathsf{rel}_F\ \overline{R}$. This relator does not satisfy IN_REL.

Example 2 (continued). By the lifted definition, $([\mathsf{Inl}\ x]_{\sim_P}, [\mathsf{Inl}\ y]_{\sim_P}) \notin \mathsf{rel}_{Q_P}\ \{\}$ because there are no (x', y') in the empty relation $\{\}$ that could be used to relate using rel_{F_P} the representatives $\mathsf{Inl}\ x'$ and $\mathsf{Inl}\ y'$. However, the witness $z = [\mathsf{Inl}\ (x, y)]_{\sim_P}$ satisfies the right-hand side of IN_REL as $\boxed{Q}\,\{\} = \{[\mathsf{Inl}\ \_]_{\sim_P}\}$. ◇

So what is the relationship between rel_Q and rel_F and under what conditions does the subdistributivity property REL_COMP hold? Like for the setter, we avoid the problematic case of empty relations by switching to $1 + \alpha$. The relator rel_{1+} adds the pair (⍟,⍟) to every relation R and thereby ensures that all relations and their compositions are non-empty. Accordingly, we obtain the following characterization:

Theorem 2 (Relator characterization).

$$([x]_\sim, [y]_\sim) \in \mathit{rel}_Q\ \overline{R} \longleftrightarrow (\mathit{map}_F\ \overline{\mathfrak{e}}\ x, \mathit{map}_F\ \overline{\mathfrak{e}}\ y) \in (\sim \bullet\ \mathit{rel}_F\ \overline{(\mathit{rel}_{1+}\ R)} \bullet \sim)$$

Moreover, the following condition on $\sim$ characterizes when rel_Q satisfies REL_COMP. Again, the non-emptiness assumptions for $R_i \bullet S_i$ come from rel_{1+} extending any relation R with the pair (⍟,⍟).

$$(\forall i.\ R_i \bullet S_i \neq \{\}) \implies \mathsf{rel}_F\ \overline{R} \bullet \sim \bullet\ \mathsf{rel}_F\ \overline{S} \subseteq \sim \bullet\ \mathsf{rel}_F\ \overline{(R \bullet S)} \bullet \sim \quad (12)$$

It turns out that this condition implies the respectfulness of the mapper (2). Intuitively, the relator is a generalization of the mapper. Furthermore, it is well known that subdistributivity implies preimage preservation [17]. Since our conditions on $\sim$ characterize these preservation properties, it is no surprise that the latter implication carries over.

Lemma 4. *Condition* (12) *implies respectfulness* (2) *and preimage preservation* (11).

In summary, we obtain the following main preservation theorem:

Theorem 3. *The quotient* $\overline{\alpha}\ Q = \overline{\alpha}\ F/\!\sim$ *inherits the structure from the BNF* $\overline{\alpha}\ F$ *with the mapper* $\mathsf{map}_Q\ \overline{f}\ [x]_\sim = [\mathsf{map}_F\ \overline{f}\ x]_\sim$ *if* $\sim$ *satisfies the conditions* (1), (9), *and* (12). *The setters and relator are given by Theorems 1 and 2, respectively.*

Example 4. A terminated coinductive list $(\alpha,\beta)\ \mathit{tllist}$ is either a finite list of α values terminated by a single β value, or an infinite list of α values. They can be seen as a quotient of pairs $\alpha\ \mathit{llist} \times \beta$, where the first component stores the possibly infinite list given by a codatatype *llist* and the second component stores the terminator. The equivalence relation identifies all pairs with the same infinite list in the first component, effectively removing the terminator from infinite lists.[2] Let $(xs,b) \sim_{\mathit{tllist}} (ys,c)$ iff $xs = ys$ and, if xs is finite, $b = c$. Like $\sim_P$ from Ex. 2, $\sim_{\mathit{tllist}}$ does not satisfy the naive condition (3).

$\texttt{codatatype}\ \ \alpha\ \mathit{llist} = \mathsf{LNil} \mid \mathsf{LCons}\ \alpha\ (\alpha\ \mathit{llist})$
$\texttt{quotient\_type}\ \ (\alpha,\beta)\ \mathit{tllist} = (\alpha\ \mathit{llist} \times \beta)/\!\sim_{\mathit{tllist}}$

Our goal is the construction of (co)datatypes with recursion through quotients such as $(\alpha,\beta)\ \mathit{tllist}$. As a realistic example, consider an inductive model of a finite interactive system that produces a possibly unbounded sequence of outputs *out* for every input *in*:

$\texttt{datatype}\ \ \mathit{system} = \mathsf{Step}\ (\mathit{in} \to (\mathit{out}, \mathit{system})\ \mathit{tllist})$

This datatype declaration is only possible if *tllist* is a BNF in β. Previously, this had to be shown by manually defining the mapper and setters and proving the BNF properties. Theorem 3 identifies the conditions under which *tllist* inherits the BNF structure of its underlying type, and it allows us to automate these definitions and proofs. For *tlllist*, the conditions can be discharged easily using automatic proof methods and a simple lemma about *llist*'s relator (stating that related lists are either both finite or infinite). ◇

3.4 Subdistributivity via confluent relations

Among the BNF properties, subdistributivity (REL_COMP) is typically the hardest to show. For example, distinct lists, type $\alpha\ \mathit{dlist}$, have been shown to be a BNF. The manual proof requires 126 lines. Of these, the subdistributivity proof takes about 100 lines. Yet, with the theory developed so far, essentially the same argument is needed for the subdistributivity condition (12). We now present a sufficient criterion for subdistributivity that simplifies such proofs. For *dlist*, this shortens the subdistributivity proof to 58 lines.

[2] Clearly, *tllist* could be defined directly as a codatatype. When Isabelle had no codatatype command, one of the authors formalized *tllist* via this quotient [25, version for Isabelle2013].

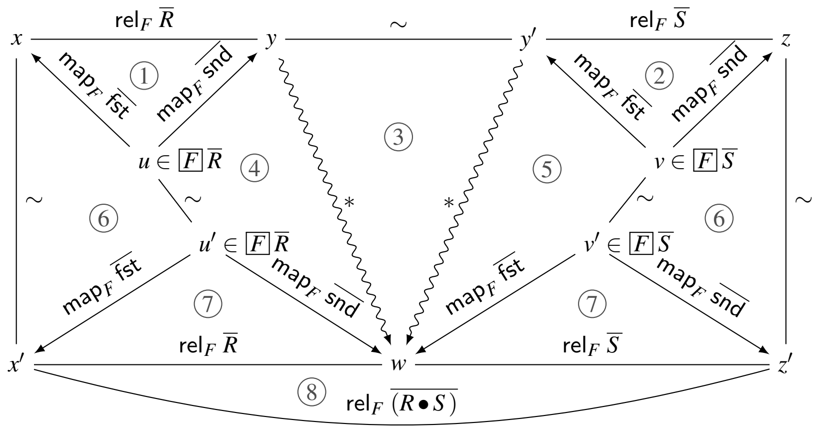
$\mathsf{rel}_F\ \overline{R}$
$\sim$
$\mathsf{rel}_F\ \overline{S}$
x
y
y'
z
$\mathsf{map}_F\ \overline{\mathsf{fst}}$
$\mathsf{map}_F\ \overline{\mathsf{snd}}$
①
②
③
④
⑤
⑥
⑦
⑧
$u \in \boxed{F}\,\overline{R}$
$v \in \boxed{F}\,\overline{S}$
$u' \in \boxed{F}\,\overline{R}$
$v' \in \boxed{F}\,\overline{S}$
$*$
x'
w
z'
$\mathsf{rel}_F\ \overline{(R \bullet S)}$

Fig. 1. Proof diagram for Theorem 4

With our `lift_bnf` command (Section 4), the whole proof is now 64 lines, half of the manual proof.

Equivalence relations are often (or can be) expressed as the equivalence closure of a rewrite relation $\rightsquigarrow$. For example, the subdistributivity proof for distinct lists views $\alpha\ \mathit{dlist}$ as the quotient $\alpha\ \mathit{list}/{\sim_{\mathit{dlist}}}$ with $\mathit{xs} \sim_{\mathit{dlist}} \mathit{ys}$ iff $\mathsf{remdups}\ \mathit{xs} = \mathsf{remdups}\ \mathit{ys}$, where $\mathsf{remdups}\ \mathit{xs}$ keeps only the last occurrence of every element in xs. So, $\sim_{\mathit{dlist}}$ is the equivalence closure of the following relation $\rightsquigarrow_{\mathit{dlist}}$, where $\cdot$ concatenates two lists:

$$\mathit{xs} \cdot [x] \cdot \mathit{ys} \rightsquigarrow_{\mathit{dlist}} \mathit{xs} \cdot \mathit{ys} \text{ if } x \in \mathsf{set}\ \mathit{ys}$$

We use the following notation: $\leftsquigarrow$ denotes the reverse relation, i.e., $x \leftsquigarrow y$ iff $y \rightsquigarrow x$. Further, $\overset{*}{\rightsquigarrow}$ denotes the reflexive and transitive closure, and $\overset{*}{\leftrightsquigarrow}$ the equivalence closure. A relation $\rightsquigarrow$ is confluent iff whenever $x \overset{*}{\rightsquigarrow} y$ and $x \overset{*}{\rightsquigarrow} z$, then there exists a u such that $y \overset{*}{\rightsquigarrow} u$ and $z \overset{*}{\rightsquigarrow} u$—or, equivalently in pointfree style, if $(\overset{*}{\leftsquigarrow} \bullet \overset{*}{\rightsquigarrow}) \subseteq (\overset{*}{\rightsquigarrow} \bullet \overset{*}{\leftsquigarrow})$.

Theorem 4 (Subdistributivity via confluent relations). *Let an equivalence relation $\sim$ satisfy* (2) *and* (3). *Then, it also satisfies* (9) *and* (12) *if there is a confluent relation $\rightsquigarrow$ with the following properties:*

(i) The equivalence relation is contained in $\rightsquigarrow$'s equivalence closure: $(\sim) \subseteq (\overset{*}{\leftrightsquigarrow})$.
(ii) The relation factors through projections: If $\mathsf{map}_F\ \overline{\mathsf{fst}}\ x \rightsquigarrow y$ then there exists a y' such that $y = \mathsf{map}_F\ \overline{\mathsf{fst}}\ y'$ and $x \sim y'$, and similarly for snd.

Proof. The wide intersection condition (9) follows from (3) by Lem. 2. The proof for the subdistributivity condition (12) is illustrated in Fig. 1. The proof starts at the top with $(x,z) \in (\mathsf{rel}_F\ \overline{R} \bullet {\sim} \bullet \mathsf{rel}_F\ \overline{S})$, i.e., there are y and y' such that $(x,y) \in \mathsf{rel}_F\ \overline{R}$ and $y \sim y'$ and $(y',z) \in \mathsf{rel}_F\ \overline{S}$. We show $(x,z) \in ({\sim} \bullet \mathsf{rel}_F\ \overline{(R \bullet S)} \bullet {\sim})$ by establishing the path from x to z via x' and z' along the three other borders of the diagram.

First ①, by IN_REL, there is a $u \in \boxed{F}\,\overline{R}$ such that $x = \mathsf{map}_F\ \overline{\mathsf{fst}}\ u$ and $y = \mathsf{map}_F\ \overline{\mathsf{snd}}\ u$. Similarly, $\mathsf{rel}_F\ \overline{S}\ y'\ z$ yields a v with the corresponding properties ②.

Second, by (i), $y \sim y'$ implies $y \overset{*}{\leftrightsquigarrow} y'$. Since $\rightsquigarrow$ is confluent, there exists a w such that $y \overset{*}{\rightsquigarrow} w$ and $y' \overset{*}{\rightsquigarrow} w$ $\textcircled{3}$. By induction on $\overset{*}{\rightsquigarrow}$ using (ii), $y \overset{*}{\rightsquigarrow} w$ factors through the projection $y = \mathsf{map}_F\ \overline{\mathsf{snd}}\ u$ and we obtain a u' such that $u \sim u'$ and $w = \mathsf{map}_F\ \overline{\mathsf{snd}}\ u'$ $\textcircled{4}$. Analogously, we obtain v' corresponding to y' and v $\textcircled{5}$. Set $x' = \mathsf{map}_F\ \overline{\mathsf{fst}}\ u'$ and $z' = \mathsf{map}_F\ \overline{\mathsf{snd}}\ v'$. As map_F preserves $\sim$ by (2), we have $x \sim x'$ and $z \sim z'$ $\textcircled{6}$.

Next, we focus on the two triangles at the bottom $\textcircled{7}$. By Lem. 2 and (3), $u \sim u'$ and $u \in \boxed{F}\,\overline{R}$ imply $u' \in \boxed{F}\,\overline{R}$; similarly $v' \in \boxed{F}\,\overline{S}$. Now, u' and v' are the witnesses to the existential in IN_REL for x' and w, and w and z', respectively. So $(x', w) \in \mathsf{rel}_F\ \overline{R}$ and $(w, z') \in \mathsf{rel}_F\ \overline{S}$, i.e., $(x', z') \in (\mathsf{rel}_F\ \overline{R} \bullet \mathsf{rel}_F\ \overline{S})$. Finally, as F is a BNF, $(x', z') \in \mathsf{rel}_F\ \overline{(R \bullet S)}$ follows with subdistributivity REL_COMP $\textcircled{8}$. □

Example 5. For distinct lists, we have $(\sim_{dlist}) = (\overset{*}{\leftrightsquigarrow}_{dlist})$ and $\rightsquigarrow_{dlist}$ is confluent. Yet, condition (ii) of Theorem 4 does not hold. For example, for $x = [(1,a),(1,b)]$, we have $\mathsf{map}_{list}\ \mathsf{fst}\ x = [1,1] \rightsquigarrow_{dlist} [1]$. However, there is no y such that $x \sim_{dlist} y$ and $\mathsf{map}_{list}\ \mathsf{fst}\ y = [1]$. The problem is that the projection $\mathsf{map}_{list}\ \mathsf{fst}$ makes different atoms of x equal and $\rightsquigarrow_{dlist}$ *removes* equal atoms, but the removal cannot be mimicked on x itself. Fortunately, we can also *add* equal atoms instead of removing them. Define $\rightsquigarrow'_{dlist}$ by

$$xs \cdot ys \rightsquigarrow'_{dlist} xs \cdot [x] \cdot ys \text{ if } x \in \mathsf{set}\ ys$$

Then, $\rightsquigarrow'_{dlist}$ is confluent and factors through projections. So distinct lists inherit the BNF structure from lists by Theorem 4. ◇

Example 6. The free monoid over atoms α consists of all finite lists α *list*. The free idempotent monoid α *fim* is then the quotient $\alpha\ list/{\sim_{fim}}$ where $\sim_{fim}$ is the equivalence closure of the idempotence law for list concatenation

$$xs \cdot ys \cdot zs \rightsquigarrow_{fim} xs \cdot ys \cdot ys \cdot zs$$

We have oriented the rule such that it introduces rather than removes the duplication. In term rewriting, the rule is typically oriented in the other direction [20] such that the resulting rewriting system terminates; however, this classical relation $\leftsquigarrow_{fim}$ is not confluent: *ababcbabc* has two normal forms $\underline{ab}\underline{abcbabc} \leftsquigarrow_{fim} \underline{abab}c \leftsquigarrow_{fim} abc$ and $\underline{abab}cbabc \leftsquigarrow_{fim} abcbabc$ (redexes are underlined). In contrast, our orientation yields a confluent relation $\rightsquigarrow_{fim}$, although the formal proof requires some effort. The relation also factors through projections. So by Theorem 4, the free idempotent monoid α *fim* is also a BNF. ◇

Example 7. A cyclic list is a finite list where the two ends are glued together. Abbot et al. [1] define the type of cyclic lists as the quotient that identifies lists whose elements have been shifted. Let $\rightsquigarrow_{rotate}$ denote the one-step rotation of a list, i.e.,

$$[] \rightsquigarrow_{rotate} [] \qquad\qquad [x] \cdot xs \rightsquigarrow_{rotate} xs \cdot [x]$$

The quotient $\alpha\ cyclist = \alpha\ list/\overset{*}{\leftrightsquigarrow}_{rotate}$ is a BNF as $\rightsquigarrow_{rotate}$ satisfies the conditions of Theorem 4. ◇

Example 1 (continued). We prove the fact that $\alpha\ re_{aci}$ is a BNF using Theorem 4. The confluent rewrite relation $\rightsquigarrow_{aci}$ that satisfies the conditions of Theorem 4 and whose equivalence closure is $\sim_{aci}$ is defined inductively as follows.

$$\begin{array}{ll}
\mathsf{Alt}\ (\mathsf{Alt}\ r\ s)\ t \leadsto_{aci} \mathsf{Alt}\ r\ (\mathsf{Alt}\ s\ t) & \mathsf{Alt}\ r\ (\mathsf{Alt}\ s\ t) \leadsto_{aci} \mathsf{Alt}\ (\mathsf{Alt}\ r\ s)\ t \\
\mathsf{Alt}\ r\ s \leadsto_{aci} \mathsf{Alt}\ s\ r & r \leadsto_{aci} \mathsf{Alt}\ r\ r \\
r \leadsto_{aci} r & \\
r \leadsto_{aci} r' \implies s \leadsto_{aci} s' \implies \mathsf{Alt}\ r\ s \leadsto_{aci} \mathsf{Alt}\ r'\ s' & \\
r \leadsto_{aci} r' \implies s \leadsto_{aci} s' \implies \mathsf{Conc}\ r\ s \leadsto_{aci} \mathsf{Conc}\ r'\ s' & \\
r \leadsto_{aci} r' \implies \mathsf{Star}\ r \leadsto_{aci} \mathsf{Star}\ r' & \diamond
\end{array}$$

4 Implementation

We provide an Isabelle/HOL command that automatically lifts the BNF structure to quotient types. The command was implemented in 1590 lines of Isabelle/ML. It requires the user to discharge our conditions on the equivalence relation. Upon success, it defines the mapper, setters, and the relator, and proves the BNF axioms and transfer rules. All automated proofs are checked by Isabelle's kernel. Eventually, the command registers the quotient type with the BNF infrastructure for use in future (co)datatype definitions.

4.1 The `lift_bnf` command

Our implementation extends the interface of the existing `lift_bnf` command for subtypes [7]. Given a quotient type $\overline{\alpha}\ Q = \overline{\alpha}\ F/{\sim}$,

`lift_bnf` $\overline{\alpha}\ Q$

asks the user to prove the conditions (9) and (12) of Theorem 3, where (9) is expressed in terms of (10) according to Lemma 3. Since the quotient construction already requires that $\sim$ be an equivalence relation, the remaining condition (1) holds trivially.

After the assumptions have been proved by the user, the command defines the BNF constants. Their definitions use an abstraction function $\mathsf{abs}_Q :: \overline{\alpha}\ F \to \overline{\alpha}\ Q$ and a representation function $\mathsf{rep}_Q :: \overline{\alpha}\ Q \to \overline{\alpha}\ F$, as in HOL Q is distinct from (but isomorphic to) the set of equivalence classes. Concretely, we define the quotient's mapper by

$$\mathsf{map}_Q\ \overline{f} = \mathsf{abs}_Q \circ \mathsf{map}_F\ \overline{f} \circ \mathsf{rep}_Q$$

The quotient's setters use the function set_{1+}, which maps $\mathfrak{e}\ a$ to $\{a\}$ and $\circledast$ to $\{\}$:

$$\mathsf{set}_{Q,i} = \Big(\lambda x.\ \bigcap_{y \in [\mathsf{map}_F\ \overline{\mathfrak{e}}\ x]_\sim} \bigcup \mathsf{set}_{1+} \langle \mathsf{set}_{F,i}\ y \rangle\Big) \circ \mathsf{rep}_Q \qquad (13)$$

This definition is equivalent to the characterization in Theorem 1. The relator (Theorem 2) is lifted similarly using rep_Q.

4.2 Transfer rule generation

The relationship of a quotient's BNF structure to its underlying type allows us to prove additional properties about the former. This is achieved by transfer rules, which drive Isabelle's Transfer tool [19] (Section 2.2). Our command automatically proves parametrized transfer rules for the lifted mapper, setters, and relator. Parametrized transfer rules are more powerful because they allow the refinement of nested types [22, Section 4.3]. They involve a parametrized correspondence relation $\mathsf{pcr}_Q\,\overline{A} = \mathsf{rel}_F\,\overline{A} \bullet \mathsf{cr}_Q$, where the parameters $\overline{A}$ relate the type arguments of F and Q.

Since map_Q is lifted canonically, its transfer rule is unsurprising:

$$(\mathsf{map}_F, \mathsf{map}_Q) \in \left(\overline{(A \Rrightarrow B)} \Rrightarrow \mathsf{pcr}_Q\,\overline{A} \Rrightarrow \mathsf{pcr}_Q\,\overline{B}\right)$$

Setters are not transferred to set_F but to the more complex function from (13):

$$\left(\lambda x.\ \bigcap\nolimits_{y \in [\mathsf{map}_F\ \overline{\mathfrak{e}}\ x]_\sim} \bigcup \mathsf{set}_{1+}\langle \mathsf{set}_{F,i}\ y\rangle,\ \mathsf{set}_{Q,i}\right) \in (\mathsf{pcr}_Q\,\overline{A} \Rrightarrow \mathsf{rel}_{\mathsf{set}}\ A_i)$$

where $(X,Y) \in \mathsf{rel}_{\mathsf{set}}\ A \longleftrightarrow (\forall x \in X.\ \exists y \in Y.\ (x,y) \in A) \wedge (\forall y \in Y.\ \exists x \in X.\ (x,y) \in A)$. Similarly, the rule for Q's relator follows its definition in Theorem 2.

Example 4 (continued). Recall that terminated coinductive lists satisfy the conditions for lifting the BNF structure. Thus, we obtain the setter $\mathsf{set}_{\mathit{tllist},2} :: (\alpha,\beta)\ \mathit{tllist} \to \beta\ \mathit{set}$ among the other BNF operations. We want to prove that $\mathsf{set}_{\mathit{tllist},2}\ x$ is empty for all infinite lists x. To make this precise, let the predicate $\mathsf{lfinite} :: \alpha\ \mathit{llist} \to \mathit{bool}$ characterize finite coinductive lists. We lift it to $(\alpha,\beta)\ \mathit{tllist}$ by projecting away the terminator:

$$\texttt{lift\_definition}\ \mathsf{tlfinite} :: (\alpha,\beta)\ \mathit{tllist} \to \mathit{bool}\ \ \texttt{is}\ \ (\lambda x.\ \mathsf{lfinite}\ (\mathsf{fst}\ x))$$

Therefore, we have to show that $\forall x.\ \neg\,\mathsf{tlfinite}\ x \implies \mathsf{set}_{\mathit{tllist},2}\ x = \{\}$. Using the transfer rules for the setter and the lifted predicate $\mathsf{tlfinite}$, the `transfer` proof method reduces the proof obligation to

$$\forall x'.\ \neg\,\mathsf{lfinite}\ (\mathsf{fst}\ x') \implies \bigcap\nolimits_{y \in [\mathsf{map}_F\ \overline{\mathfrak{e}}\ x']_{\sim_{\mathit{tllist}}}} \bigcup \mathsf{set}_{1+}\langle \mathsf{set}_{F,2}\ y\rangle = \{\}$$

where $x' :: (\alpha,\beta)\ F$, and $(\alpha,\beta)\ F = (\alpha\ \mathit{llist} \times \beta)$ is the underlying functor of *tllist*. The rest of the proof, which need not refer to *tllist* anymore, is automatic. ◇

We have also extended `lift_bnf` to generate transfer rules for subtypes. There, the setters and the relator do not change: if T is a subtype of F, e.g., then $\mathsf{set}_{T,i}$ is transferred to $\mathsf{set}_{F,i}$.

5 Related work

Quotient constructions have been formalized and implemented, e.g., in Isabelle/HOL [19, 21, 34, 35], HOL4 [18], Agda [40, 41], Cedille [30], Coq [11, 12], Lean [3], and Nuprl [33]. None of these works look at the preservation of functor properties except for Avigad et al. [3] (discussed in Section 5.2) and Veltri [41]. Veltri studies the special case of when the delay monad is preserved by a quotient of weak bisimilarity, focusing on the challenges that quotients pose in intensional type theory.

Abbot et al. [1] introduce quotient containers as a model of datatypes with permutative structure, such as unordered pairs, cyclic lists, and multisets. The map function of quotient containers does not change the shape of the container. Quotient containers therefore cannot deal with quotients where the equivalence relation takes the identity of elements into account, such as distinct lists, finite sets, and the free idempotent monoid. Overall our construction strictly subsumes quotient containers.

5.1 Quotients in the category of Sets

BNFs are accessible functors in the category of Sets. We therefore relate to the literature on when quotients preserve functors and their properties in Set.

Trnková [38] showed that all Set functors preserve non-empty intersections: in our notation $\boxed{F}\,A \cap \boxed{F}\,B = \boxed{F}\,(A \cap B)$ whenever $A \cap B \neq \{\}$. Empty intersections need not be preserved though. Functors that do are called regular [39] or sound [2]. All BNFs are sound as $\boxed{F}\,A = \{x \mid \mathsf{set}_F\ x \subseteq A\}$. As shown in Example 2, the naive quotient construction can lead to unsound functors.

Every unsound functor can be "repaired" by setting $\boxed{F}\,\{\}$ to the *distinguished points* dp_F. We write $\boxed{F}'$ for the repaired action.

$$\boxed{F}'\,A = \begin{cases} \mathsf{dp}_F & \text{if } A = \{\} \\ \boxed{F}\,A & \text{otherwise} \end{cases} \tag{14}$$

Trnková characterizes the distinguished points dp_F as the natural transformations from $C_{1,0}$ to F where $\boxed{C_{1,0}}\,\{\} = \{\}$ and $\boxed{C_{1,0}}\,A = \{\circledast\}$ for $A \neq \{\}$. Barr [4] and Gumm [16] use equalizers instead of natural transformations to define the distinguished points of univariate functors:

$$\mathsf{dp}_F = \{x \mid \mathsf{map}_F\ (\lambda\_.\ \mathsf{True})\ x = \mathsf{map}_F\ (\lambda\_.\ \mathsf{False})\ x\} \tag{15}$$

The case distinction in (14) makes it hard to work with repaired functors, especially as the case distinctions proliferate for multivariate functors. Instead, we repair the unsoundness by avoiding empty sets altogether: Our characterization $\colorbox{gray}{$F$}\,A$ in Lemma 1 effectively derives the quotient from $(1+\alpha)\ F$ instead of $\alpha\ F$. Moreover, our characterization of $\boxed{F}\,\overline{A}$ generalizes Barr and Gumm's definition of distinguished points: for $\overline{A} = \{\}$, (5) simplifies to (15). The resulting quotient is the same because $[\colorbox{gray}{$F$}\,\overline{A}]_\sim = [\boxed{F}\,\overline{A}]_\sim$ if $A_i \neq \{\}$ for all i.

Given the other BNF properties, subdistributivity is equivalent to the functor preserving weak pullbacks. Adámek et al. [2] showed that an accessible Set functor preserves weak pullbacks iff it has a so-called dominated presentation in terms of flat equations E over a signature Σ. This characterization does not immediately help with proving subdistributivity, though. For example, the finite set quotient $\alpha\ \mathit{fset} = \alpha\ \mathit{list}/\!\sim_{\mathit{fset}}$ comes with the signature $\Sigma = \{\sigma_n \mid n \in \mathbb{N}\}$ and the equations $\sigma_n(x_1, \dots x_n) = \sigma_m(y_1, \dots, y_m)$ whenever $\{x_1, \dots, x_n\} = \{y_1, \dots, y_m\}$. Proving domination for this presentation boils down to proving subdistributivity directly. Our criterion using a confluent relation (Theorem 4) is only sufficient, not necessary, but it greatly simplifies the actual proof effort.

5.2 Comparison with Lean's quotients of polynomial functors

Avigad et al. [3] proposed quotients of polynomial functors (QPF) as a model for datatypes. QPFs generalize BNFs in that they require less structure: there is no setter and the relator need not satisfy subdistributivity. Nevertheless, the quotient construction is similar to ours. Without loss of generality, we consider in our comparison only the univariate case $\alpha\ Q = \alpha\ F/\!\sim$.

The main difference lies in the definition of the liftings lift_F of predicates $P :: \alpha \to \mathbb{B}$ and relations $R :: \alpha \otimes \beta$. In our notation, $\mathsf{lift}_F\ P$ corresponds to $\lambda x.\ x \in \boxed{F}\ \{a \mid P\ a\}$ and $\mathsf{lift}_F\ R$ to $\mathsf{rel}_F\ R$. QPF defines these liftings for the quotient Q as follows:

$$\mathsf{lift}_Q\ P\ [x]_\sim = (\exists x' \in [x]_\sim.\ P\ x') \qquad \mathsf{lift}_Q\ R\ [x]_\sim\ [y]_\sim = (\exists x' \in [x]_\sim.\ \exists y' \in [y]_\sim.\ R\ x'\ y')$$

That is, these definitions correspond to the naive construction $\boxed{Q}\,A = [\boxed{F}\,A]_\sim$ and $\mathsf{rel}_Q\ R = [\mathsf{rel}_F\ R]_\sim$ where $[(x,y)]_\sim = ([x]_\sim,[y]_\sim)$. As discussed above, the resulting quotient may be an unsound functor. Consequently, lifting of predicates does not preserve empty intersections in general. This hinders modular proofs. For example, suppose that a user has already shown $\mathsf{lift}_Q\ P_1\ x$ and $\mathsf{lift}_Q\ P_2\ x$ for some value x and two properties P_1 and P_2. Then, to deduce $\mathsf{lift}_F\ (\lambda a.\ P_1\ a \wedge P_2\ a)\ x$, they would have to prove that the two properties do not contradict each other, i.e., $\exists a.\ P_1\ a \wedge P_2\ a$. Obviously, this makes modular proofs harder as extra work is needed to combine properties.

QPF uses $\mathsf{lift}_F\ P$ in the induction theorem for datatypes. So when a datatype recurses through *tllist*, this spreads to proofs by induction: splitting a complicated inductive statement into smaller lemmas is not for free. Moreover, lift_Q holds for fewer values, as the next example shows. Analogous problems arise in QPF for relation lifting, which appears in the coinduction theorem.

Example 4 (continued). Consider the infinite repetition $\mathsf{repeat}\ a :: (\alpha,\beta)\ \mathit{tllist}$ of the atom a as a terminated lazy list. As $\mathsf{repeat}\ a$ contains only as, one would expect that $\mathsf{lift}_{\mathit{tllist}}\ (\lambda a'.\ a' = a)\ (\lambda\_.\ \mathsf{False})\ (\mathsf{repeat}\ a)$ holds. Yet, this property is provably false. $\diamond$

These issues would go away if lift_Q was defined following our approach for $\boxed{Q}\,A = [\boxed{F}\,A]_\sim$ and rel_Q as in Theorem 2. These definitions do not rely on the additional BNF structure; only map_Q is needed and QPF defines map_Q like we do. The repair should therefore work for the general QPF case, as well.

6 Conclusion

We have described a sufficient criterion for quotient types to be able to inherit the BNF structure from the underlying type. We have demonstrated the effectiveness of the criterion by automating the BNF "inheritance" in the form of the `lift_bnf` command in Isabelle/HOL and used it (which amounts to proving the criterion) for several realistic quotient types. We have also argued that our treatment of the quotient's setter and relator to avoid unsoundness carries over to more general structures, such as Lean's QPFs.

As future work, we plan to investigate quotients of existing generalizations of BNFs to co- and contravariant functors [28] and functors operating on small-support endomorphisms and bijections [8]. Furthermore, we would like to provide better automation for proving subdistributivity via confluent rewrite systems as part of `lift_bnf`.

Acknowledgment We thank David Basin for supporting this work, Ralf Sasse and Andrei Popescu for insightful discussions about confluent relations, BNFs, their preservation of wide intersections, and ways to express the setters in terms of the mapper, and Jasmin Blanchette and the anonymous IJCAR reviewers for numerous comments on earlier drafts of this paper, which helped to improve the presentation. Julian Biendarra developed the `lift_bnf` command for subtypes, which we extend to quotient types in this work.

References

1. M. Abbott, T. Altenkirch, N. Ghani, and C. McBride. Constructing polymorphic programs with quotient types. In D. Kozen, editor, *MPC 2004*, volume 3125 of *LNCS*, pages 2–15, Berlin, Heidelberg, 2004. Springer.
2. J. Adámek, H. P. Gumm, and V. Trnková. Presentation of set functors: A coalgebraic perspective. *J. Log. Comput.*, 20(5):991–1015, 2010.
3. J. Avigad, M. Carneiro, and S. Hudon. Data types as quotients of polynomial functors. In J. Harrison, J. O'Leary, and A. Tolmach, editors, *ITP 2019*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
4. M. Barr. Terminal coalgebras in well-founded set theory. *Theor. Comput. Sci.*, 114(2):299–315, 1993.
5. D. A. Basin, S. Krstic, and D. Traytel. Almost event-rate independent monitoring of metric dynamic logic. In S. K. Lahiri and G. Reger, editors, *RV 2017*, volume 10548 of *LNCS*, pages 85–102. Springer, 2017.
6. D. A. Basin, A. Lochbihler, and S. R. Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *J. Cryptology*, 33:494–566, 2020.
7. J. Biendarra. Functor-preserving type definitions in Isabelle/HOL. Bachelor thesis, Fakultät für Informatik, Technische Universität München, 2015.
8. J. C. Blanchette, L. Gheri, A. Popescu, and D. Traytel. Bindings as bounded natural functors. *PACMPL*, 3(POPL):22:1–22:34, 2019.
9. J. C. Blanchette, J. Hölzl, A. Lochbihler, L. Panny, A. Popescu, and D. Traytel. Truly modular (co)datatypes for Isabelle/HOL. In G. Klein and R. Gamboa, editors, *ITP 2014*, volume 8558 of *LNCS*, pages 93–110. Springer, 2014.
10. J. C. Blanchette, A. Popescu, and D. Traytel. Cardinals in Isabelle/HOL. In G. Klein and R. Gamboa, editors, *ITP 2014*, volume 8558 of *LNCS*, pages 111–127. Springer, 2014.
11. L. Chicli, L. Pottier, and C. Simpson. Mathematical quotients and quotient types in Coq. In H. Geuvers and F. Wiedijk, editors, *TYPES 2003*, volume 2646 of *LNCS*, pages 95–107. Springer, 2003.
12. C. Cohen. Pragmatic quotient types in Coq. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP 2013*, volume 7889 of *LNCS*, pages 213–228. Springer, 2013.
13. C. Cohen, M. Dénès, and A. Mörtberg. Refinements for free! In G. Gonthier and M. Norrish, editors, *CPP 2013*, volume 8307 of *LNCS*, pages 147–162. Springer, 2013.
14. G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In F. Rossi, editor, *IJCAI 2013*, pages 854–860. IJCAI/AAAI, 2013.
15. B. Fürer, A. Lochbihler, J. Schneider, and D. Traytel. Quotients of bounded natural functors (extended report). Technical report, 2020. `https://people.inf.ethz.ch/trayteld/papers/qbnf/qbnf_report.pdf`.
16. H. P. Gumm. From T-coalgebras to filter structures and transition systems. In J. L. Fiadeiro, N. Harman, M. Roggenbach, and J. Rutten, editors, *Algebra and Coalgebra in Computer Science*, volume 3629 of *LNCS*, pages 194–212, Berlin, Heidelberg, 2005. Springer.
17. H. P. Gumm and T. Schröder. Types and coalgebraic structure. *Algebra universalis*, 53(2):229–252, 2005.
18. P. V. Homeier. A design structure for higher order quotients. In J. Hurd and T. Melham, editors, *TPHOLs 2005*, volume 3603 of *LNCS*, pages 130–146, Berlin, Heidelberg, 2005. Springer.
19. B. Huffman and O. Kunčar. Lifting and Transfer: A modular design for quotients in Isabelle/HOL. In G. Gonthier and M. Norrish, editors, *CPP 2013*, volume 8307 of *LNCS*, pages 131–146. Springer, 2013.

20. J.-M. Hullot. A catalogue of canonical term rewrite systems. Technical Report CSL-113, SRI International, 1980.
21. C. Kaliszyk and C. Urban. Quotients revisited for Isabelle/HOL. In W. C. Chu, W. E. Wong, M. J. Palakal, and C. Hung, editors, *SAC 2011*, pages 1639–1644. ACM, 2011.
22. O. Kunčar. *Types, Abstraction and Parametric Polymorphism in Higher-Order Logic*. PhD thesis, Technical University Munich, Germany, 2016.
23. O. Kunčar and A. Popescu. From types to sets by local type definition in higher-order logic. *J. Autom. Reasoning*, 62(2):237–260, 2019.
24. P. Lammich and A. Lochbihler. Automatic refinement to efficient data structures: A comparison of two approaches. *J. Autom. Reasoning*, 63(1):53–94, 2019.
25. A. Lochbihler. Coinductive. *Archive of Formal Proofs*, 2010. `http://isa-afp.org/entries/Coinductive.html`, Formal proof development.
26. A. Lochbihler. Effect polymorphism in higher-order logic (proof pearl). *J. Autom. Reasoning*, 63(2):439–462, 2019.
27. A. Lochbihler and J. Schneider. Equational reasoning with applicative functors. In J. C. Blanchette and S. Merz, editors, *ITP 2016*, volume 9807 of *LNCS*, pages 252–273. Springer, 2016.
28. A. Lochbihler and J. Schneider. Relational parametricity and quotient preservation for modular (co)datatypes. In J. Avigad and A. Mahboubi, editors, *ITP 2018*, volume 10895 of *LNCS*, pages 411–431. Springer, 2018.
29. A. Lochbihler, S. R. Sefidgar, D. A. Basin, and U. Maurer. Formalizing constructive cryptography using CryptHOL. In *CSF 2019*, pages 152–166. IEEE, 2019.
30. A. Marmaduke, C. Jenkins, and A. Stump. Quotient types by normalization in Cedille. In *TFP 2019*, 2019.
31. T. Nipkow and G. Klein. *Concrete Semantics - With Isabelle/HOL*. Springer, 2014.
32. T. Nipkow and D. Traytel. Unified decision procedures for regular expression equivalence. In G. Klein and R. Gamboa, editors, *ITP 2014*, volume 8558 of *LNCS*, pages 450–466. Springer, 2014.
33. A. Nogin. Quotient types: A modular approach. In V. A. Carreño, C. A. Muñoz, and S. Tahar, editors, *TPHOLs 2002*, volume 2410 of *LNCS*, pages 263–280. Springer, 2002.
34. L. C. Paulson. Defining functions on equivalence classes. *ACM Trans. Comput. Logic*, 7(4):658–675, 2006.
35. O. Slotosch. Higher order quotients and their implementation in Isabelle/HOL. In E. L. Gunter and A. Felty, editors, *TPHOLs 1997*, volume 1275 of *LNCS*, pages 291–306, Berlin, Heidelberg, 1997. Springer.
36. M. Sozeau. A new look at generalized rewriting in type theory. *J. Formalized Reasoning*, 2(1):41–62, 2010.
37. D. Traytel, A. Popescu, and J. C. Blanchette. Foundational, compositional (co)datatypes for higher-order logic: Category theory applied to theorem proving. In *LICS 2012*, pages 596–605. IEEE Computer Society, 2012.
38. V. Trnková. Some properties of set functors. *Commentationes Mathematicae Universitatis Carolinae*, 10(2):323–352, 1969.
39. V. Trnková. On descriptive classification of set-functors I. *Commentationes Mathematicae Universitatis Carolinae*, 12(1):143–174, 1971.
40. N. Veltri. Two set-based implementations of quotients in type theory. In J. Nummenmaa, O. Sievi-Korte, and E. Mäkinen, editors, *SPLST 2015*, volume 1525 of *CEUR Workshop Proceedings*, pages 194–205, 2015.
41. N. Veltri. *A Type-Theoretical Study of Nontermination*. PhD thesis, Tallinn University of Technology, 2017.