Relational Parametricity and Quotient Preservation for Modular (Co)datatypes

Andreas Lochbihler and Joshua Schneider

Institute of Information Security, Department of Computer Science, ETH Zürich, Zürich, Switzerland
andreas.lochbihler@inf.ethz.ch, joshua.schneider@inf.ethz.ch

Abstract. Bounded natural functors (BNFs) provide a modular framework for the construction of (co)datatypes in higher-order logic. Their functorial operations, the mapper and relator, are restricted to a subset of the parameters, namely those where recursion can take place. For certain applications, such as free theorems, data refinement, quotients, and generalised rewriting, it is desirable that these operations do not ignore the other parameters. In this paper, we generalise BNFs such that the mapper and relator act on both covariant and contravariant parameters. Our generalisation, $\mathrm{BNF}_{\mathrm{CC}}$, is closed under functor composition and least and greatest fixpoints. In particular, every (co)datatype is a $\mathrm{BNF}_{\mathrm{CC}}$. We prove that subtypes inherit the $\mathrm{BNF}_{\mathrm{CC}}$ structure under conditions that generalise those for the BNF case. We also identify sufficient conditions under which a $\mathrm{BNF}_{\mathrm{CC}}$ preserves quotients. Our development is formalised abstractly in Isabelle/HOL in such a way that it integrates seamlessly with the existing parametricity infrastructure.

1 Introduction

Datatypes and codatatypes are a fundamental tool in functional programming and proof assistants. Proof assistants based on type theory usually provide (co)datatypes as a built-in concept (e.g. Coq [31], Agda [29], Lean [28]), whereas other tools defer the construction to definitional (Isabelle [7], HOL [36]) or axiomatic packages (PVS [30], Dafny [23]). Traytel et al. [38] proposed *bounded natural functors* (BNFs) as a semantic criterion for (co)datatypes that are constructible in higher-order logic, which was subsequently implemented as a definitional package in Isabelle/HOL [7]. Notably, the BNF class includes important non-free types such as finite sets and discrete probability distributions. The package allows a modular approach: Once a type constructor has been proved to be a BNF, it can be used to define new (co)datatypes.

For example, following the coalgebraic theory of systems [34], deterministic discrete systems are modelled as elements of a codatatype $(\mathfrak{a}, \mathfrak{b})\ \mathsf{dds}$, where $\mathfrak{a}$ is the type of inputs, and $\mathfrak{b}$ is the type of outputs of the system. In Isabelle/HOL, the following command defines this type with constructor Dds and destructor run.

$$\mathsf{codatatype}\ (\mathfrak{a}, \mathfrak{b})\ \mathsf{dds} = \mathsf{Dds}\ (\mathsf{run}\colon \mathfrak{a} \Rightarrow \mathfrak{b} \times (\mathfrak{a}, \mathfrak{b})\ \mathsf{dds})$$

Note that $(\mathfrak{a}, \mathfrak{b})\ \mathsf{dds}$ on the right-hand side occurs inside a product $(\mathfrak{b} \times \square)$ and a function type $(\mathfrak{a} \Rightarrow \square)$, which both are BNFs. Yet, not all recursive specifications produce valid HOL (co)datatypes [13]. For example, a datatype must not recurse through the domain of predicates $\square \Rightarrow \mathsf{bool}$. Otherwise, HOL's set-theoretic semantics would have to contain an

injection into a non-empty set from its powerset. To avoid such inconsistencies, BNFs distinguish *live* type parameters from *dead* ones, and (co)recursion is limited to live parameters. For the function space $\mathfrak{a} \Rightarrow \mathfrak{b}$, $\mathfrak{b}$ is live and $\mathfrak{a}$ is dead, and the same holds for $(\mathfrak{a}, \mathfrak{b})\ \mathsf{dds}$.

Many type constructors come with a map operation (mapper) that lifts unary functions on the type parameters to the whole type. For lists, e.g., the mapper $\mathsf{map}_{\mathsf{list}} :: (\mathfrak{a} \Rightarrow \mathfrak{b}) \Rightarrow \mathfrak{a}\ \mathsf{list} \Rightarrow \mathfrak{b}\ \mathsf{list}$ applies the given function to all elements in the list. The function space's mapper $\mathsf{map}_{\Rightarrow}\ g\ h\ f = h \circ f \circ g$ transforms both the domain and the range of f. Every BNF has a mapper, but it acts only on the live parameters. For the function space, the BNF mapper $\mathsf{map}_{\Rightarrow}\ \mathsf{id}\ h$ therefore transforms only the range, but not the domain. Similarly, the BNF mapper $\mathsf{map}_{\mathsf{dds}}$ for DDS's has type $(\mathfrak{b} \Rightarrow \mathfrak{b}') \Rightarrow (\mathfrak{a}, \mathfrak{b})\ \mathsf{dds} \Rightarrow (\mathfrak{a}, \mathfrak{b}')\ \mathsf{dds}$, i.e., it can transform a system's outputs. But it is useless if we want to transform the inputs. For example, consider a system S turning integers into booleans (e.g., testing whether the partial sum of inputs is even). Then, we cannot easily use it on natural numbers. In contrast, if the mapper acted also on the contravariant type parameter $\mathfrak{a}$, i.e., $\mathsf{map}_{\mathsf{dds}} :: (\mathfrak{a}' \Rightarrow \mathfrak{a}) \Rightarrow (\mathfrak{b} \Rightarrow \mathfrak{b}') \Rightarrow (\mathfrak{a}, \mathfrak{b})\ \mathsf{dds} \Rightarrow (\mathfrak{a}', \mathfrak{b}')\ \mathsf{dds}$, then the new system could be written as $\mathsf{map}_{\mathsf{dds}}\ \mathsf{int}\ \mathsf{id}\ S$, where $\mathsf{int} :: \mathsf{nat} \Rightarrow \mathsf{int}$ embeds the natural numbers in the integers.

This limitation of the BNF mapper is pervasive. First of all, it also affects the derived relator, which lifts relations rather than functions. For example, the list relator $\mathsf{rel}_{\mathsf{list}}\ R$ relates two lists iff they have the same length and the elements at the same indices are related by R. The function space relator $A \Rightarrow B$ takes a relation A on the domain and a relation B on the codomain. It relates two functions if they map A-related inputs to B-related outputs. But when seen as a BNF, A is always fixed to the identity relation $(=)$. Accordingly, due to the modular construction of (co)datatypes, the DDS relator lifts only a relation on outputs, and the input's relation is fixed to $(=)$.

Mappers and relators are used by many reasoning tools built on relational parametricity [33]. A polymorphic term is relationally parametric if its instances for related types are related, too. This requires an interpretation of types as relations and, thus, relators. The BNF restriction to live parameters hampers many applications of the interpretation: Quotients cannot be lifted through dead parameters [17], data refinement cannot happen in dead parameter positions [9,21], rewriting with equivalence relations must not affect dead parameters [37], free theorems can talk only about live parameters [39], and so on. Whenever—today in Isabelle/HOL—any of this is needed for dead parameters, too, one has to manually define more general mappers and relators ad hoc for the affected types.

In this paper, we generalise the BNF notion to BNF_{CC}, where dead parameters are refined into covariant, contravariant, and fixed parameters—"CC" stands for covariance and contravariance. While live parameters are technically covariant, we reserve the latter term for non-live parameters. For example, the type of second-order functions $(\mathfrak{a} \Rightarrow \mathfrak{b}) \Rightarrow \mathfrak{c}$ is a BNF where only $\mathfrak{c}$ is live and $\mathfrak{a}$ and $\mathfrak{b}$ are dead. Considered as a BNF_{CC}, $\mathfrak{c}$ is live, $\mathfrak{b}$ is contravariant because it occurs at a negative position with respect to the function space, and $\mathfrak{a}$ is covariant as it occurs at a positive, but not strictly positive position. The BNF_{CC} mapper and relator act on all type parameters but the fixed ones. For dds, e.g., we do obtain the desired mapper that lets us transform both the inputs and the outputs. The BNF_{CC} notion coincides with the BNF notion when there are only live and fixed parameters.

The key feature of BNFs is that they are closed under composition and least and greatest fixpoints, i.e., (co)datatypes. BNF_{CC}s also enjoy these properties. So, they are

as modular as the BNFs and can be integrated into Isabelle's (co)datatype package. We emphasise that BNF_{CC}s do not allow users to define more (co)datatypes than BNFs do. The difference is that BNF_{CC}s yield more versatile mappers and relators with useful properties. Moreover, they integrate nicely with the rest of the functor-based infrastructure.

The main contributions of this paper are the following:

- We introduce the notion of BNF_{CC} as a generalisation of BNF (Sect. 2). BNF_{CC}s are equipped with more general relators and mappers than what the underlying natural functor provides. These operations are useful for various reasoning tasks (Sect. 1.2).
- We prove that BNF_{CC}s are closed under composition (Sect. 3) and least and greatest fixpoints (Sect. 4). This makes BNF_{CC}s modular and avoids syntactic conditions on definitions. In particular, every (co)datatype defined with Isabelle/HOL's (co)datatype package [7,8] is a BNF_{CC}.
- We prove that subtypes preserve the BNF_{CC} structure under certain conditions (Sect. 5). Consequently, non-uniform (co)datatypes [8] are BNF_{CC}s, too. If there are no covariant and contravariant parameters, our conditions are equivalent to those for BNFs.
- We prove that BNF_{CC}s lift quotients unconditionally through live parameters and under mild conditions through covariant and contravariant parameters (Sect. 6). This makes Isabelle's Lifting package more powerful, as the BNF theory only proves lifting for live parameters.

We formalised all our constructions and proofs in Isabelle/HOL [25]. Since reasoning about abstract functors is impossible in HOL, we axiomatised two generic BNF_{CC}s with sufficiently many parameters of each kind, and used them for the concrete constructions. The formalisation includes the examples from Sections 1 and 2. In addition, we give informal proof sketches for most propositions and theorems in Appendix A (Online Resource). The implementation of BNF_{CC}s as an extension of the existing packages is left as future work (Sect. 8).

1.1 Background: Bounded Natural Functors

A bounded natural functor (BNF) [38] is a type constructor F of some arity equipped with a mapper, conversions to sets, and a cardinality bound on those sets. A type parameter of F is either *live* or *dead*; dead parameters are ignored by the BNF operations. The mapper is given by the polymorphic operation $\mathsf{map}_{\mathsf{F}} :: \overline{(\mathfrak{l} \Rightarrow \mathfrak{l}')} \Rightarrow (\overline{\mathfrak{l}}, \overline{\mathfrak{d}})\ \mathsf{F} \Rightarrow (\overline{\mathfrak{l}'}, \overline{\mathfrak{d}})\ \mathsf{F}$ on the live parameters $\overline{\mathfrak{l}}$, whereas the dead parameters $\overline{\mathfrak{d}}$ remain fixed.[1] We assume without loss of generality that all live parameters precede the dead ones in the parameter list. For each live type parameter $\mathfrak{l}_i$, a BNF comes with a polymorphic setter $\mathsf{set}^i_{\mathsf{F}} :: (\overline{\mathfrak{l}}, \overline{\mathfrak{d}})\ \mathsf{F} \Rightarrow \mathfrak{l}_i\ \mathsf{set}$. The cardinality bound bd_{F} is assumed to be infinite and may depend only on non-live parameters. The BNF operations must satisfy the following laws [7]:

[1] The notation $\bar{x}$ stands for a meta-syntactic list of formal entities $x_1, x_2, \ldots, x_n$. We use this notation quite liberally, such that the expanded type of $\mathsf{map}_{\mathsf{F}}$ reads

$$(\mathfrak{l}_1 \Rightarrow \mathfrak{l}'_1) \Rightarrow (\mathfrak{l}_2 \Rightarrow \mathfrak{l}'_2) \Rightarrow \ldots \Rightarrow (\mathfrak{l}_m \Rightarrow \mathfrak{l}'_m) \Rightarrow (\mathfrak{l}_1, \ldots, \mathfrak{l}_m, \mathfrak{d}_1, \ldots, \mathfrak{d}_n)\ \mathsf{F} \Rightarrow (\mathfrak{l}'_1, \ldots, \mathfrak{l}'_m, \mathfrak{d}_1, \ldots, \mathfrak{d}_n)\ \mathsf{F}.$$

Similarly, we write $\forall i.\ \varphi$ for the conjunction of all instances of φ over the index i. Superscripts select a subsequence, e.g., $\overline{x^{>2}}$ represents $x_3, x_4, \ldots, x_n$.

$$\mathsf{map_F}\ \overline{\mathsf{id}} = \mathsf{id} \qquad \mathsf{map_F}\ \overline{(f \circ g)} = \mathsf{map_F}\ \overline{f} \circ \mathsf{map_F}\ \overline{g} \qquad \forall i.\ |\mathsf{set}^i_{\mathsf{F}}| \leq \mathsf{bd_F} \quad (1)$$

$$\forall i.\ \mathsf{set}^i_{\mathsf{F}}\ (\mathsf{map_F}\ \overline{f}\ x) = f_i\ `\mathsf{set}^i_{\mathsf{F}}\ x \qquad \frac{\forall i.\ \forall y \in \mathsf{set}^i_{\mathsf{F}}\ x.\ f_i\ y = g_i\ y}{\mathsf{map_F}\ \overline{f}\ x = \mathsf{map_F}\ \overline{g}\ x} \quad (2)$$

$$\mathsf{rel_F}\ \overline{R} \bigcirc \mathsf{rel_F}\ \overline{S} \sqsubseteq \mathsf{rel_F}\ \overline{(R \bigcirc S)} \quad (3)$$

Here, $f\ `A = \{y \mid \exists x \in A.\ y = f\ x\}$ denotes A's image under f, $|A|$ is A's cardinality, $\bigcirc$ is relation composition, and $\sqsubseteq$ is relation containment. Relations are represented as binary predicates of type $\mathfrak{a} \otimes \mathfrak{b} = (\mathfrak{a} \Rightarrow \mathfrak{b} \Rightarrow \mathsf{bool})$. The relator $\mathsf{rel_F}$ is defined as

$$\mathsf{rel_F}\ \overline{R}\ x\ y = (\exists z.\ (\forall i.\ \mathsf{set}^i_{\mathsf{F}}\ z \subseteq \{(a,b) \mid R_i\ a\ b\}) \wedge \mathsf{map_F}\ \overline{\pi_1}\ z = x \wedge \mathsf{map_F}\ \overline{\pi_2}\ z = y) \quad (4)$$

where π_1 and π_2 project a pair to its components. The relator extends $\mathsf{map_F}$ to relations, interpreting functions f by their graphs $\mathsf{Gr}\ f = (\lambda x\ y.\ y = f\ x)$:

Lemma 1. *If F is a BNF, then* $\mathsf{Gr}\ (\mathsf{map_F}\ \overline{f}) = \mathsf{rel_F}\ \overline{(\mathsf{Gr}\ f)}$.

BNFs are closed under various operations: functor composition, "killing" of live type parameters, and least and greatest fixpoints. Examples of basic BNFs are the identity functor, products ($\times$), sums ($+$), and function spaces ($\Rightarrow$), where the domain is dead. Finite lists $\mathfrak{a}$ list are a datatype and hence a BNF, too, with mapper $\mathsf{map_{list}}$, relator $\mathsf{rel_{list}}$, and bound $\aleph_0$. The setter $\mathsf{set_{list}}$ returns the set of elements in the list.

1.2 Examples and Applications

We now illustrate the benefits of parametricity-based reasoning using small examples, which all require the generalised mappers and relators. Although all our examples revolve around the DDS codatatype, parametricity-based reasoning is not restricted to coalgebraic system models. It can equally be used for all the other (co)datatypes, and whenever a type parameter is covariant or contravariant (e.g., $\mathfrak{a}$ in $(\mathfrak{a},\mathfrak{b})$ tree $=$ Leaf $\mathfrak{b}$ | Node $(\mathfrak{a} \Rightarrow (\mathfrak{a},\mathfrak{b})$ tree$)$), the $\mathrm{BNF_{CC}}$ theory makes the reasoning more powerful than the BNF theory.

Free theorems. Wadler [39] showed how certain theorems can be derived from parametricity by instantiating the relations with the graphs of functions and using Lemma 1, which we generalise to $\mathrm{BNF_{CC}}$s in Sect. 2. As shown in the introduction, the inputs and outputs of a DDS can be transformed with the mapper $\mathsf{map_{dds}}$. Parallel $||$ and sequential $\bullet$ composition for DDS's, e.g., are defined corecursively by

```
primcorec (||) :: (𝔞,𝔟) dds ⇒ (𝔠,𝔡) dds ⇒ (𝔞+𝔠,𝔟+𝔡) dds where
  run (S1 || S2) = (λx. case x of
      Inl a ⇒ let (b,S1') = run S1 a in (Inl b,S1' || S2)
    | Inr c ⇒ let (d,S2') = run S2 c in (Inr d,S1 || S2'))
primcorec (•) :: (𝔞,𝔟) dds ⇒ (𝔟,𝔠) dds ⇒ (𝔞,𝔠) dds where
  run (S1 • S2) = (λa. let (b,S1') = run S1 a; (c,S2') = run S2 b in (c,S1' • S2'))
```

where Inl and Inr denote the injections into the sum type.

The following "free" theorems are derived from the parametricity laws by rewriting only; no coinduction is needed. Note that the BNF mapper on live parameters only would not be any good for $\bullet$ as the function g occurs both in the live and dead positions.

$$\mathsf{map}_{\mathsf{dds}}\ f\ h\ S_1 \,||\, \mathsf{map}_{\mathsf{dds}}\ g\ k\ S_2 = \mathsf{map}_{\mathsf{dds}}\ (\mathsf{map}_+\ f\ g)\ (\mathsf{map}_+\ h\ k)\ (S_1 \,||\, S_2)$$
$$\mathsf{map}_{\mathsf{dds}}\ f\ g\ S_1 \bullet S_2 = \mathsf{map}_{\mathsf{dds}}\ f\ \mathsf{id}\ (S_1 \bullet \mathsf{map}_{\mathsf{dds}}\ g\ \mathsf{id}\ S_2)$$
$$S_1 \bullet \mathsf{map}_{\mathsf{dds}}\ g\ h\ S_2 = \mathsf{map}_{\mathsf{dds}}\ \mathsf{id}\ h\ (\mathsf{map}_{\mathsf{dds}}\ \mathsf{id}\ g\ S_1 \bullet S_2)$$

Reasoning with parametricity is especially useful in larger applications. The first author formalised a cryptographic algebra based on sub-probabilistic discrete systems (sPDS) similar to Maurer's random systems [26]. Deriving the free theorems from parametricity pays off particularly for transformers of sPDS, which are formalised as a codatatype that recurses through another codatatype of probabilistic resumptions. Proofs by coinduction would require substantially more effort even for such simple theorems.

Data refinement. Data refinement changes the representation of data in a program. It offers a convenient way to go from abstract data structures like sets to efficient ones like red-black trees, which are the key to generate efficient code from a formalisation. Several tools automate the data refinement and synthesise an implementation from an abstract specification in this way [9,10,14,21]. As these tools are based on parametricity, (nested) data refinement is only possible in type parameters on which the relators act. A more general relator thus increases the refinement capabilities.

For example, consider a DDS traverse G parametrised by a finite graph G. Upon input of a node set A, it returns all successor nodes $G[A]$ of A that have not yet been visited. Such a DDS can be used to implement a breadth-first or depth-first search traversal of a graph. Suppose that the correctness proof works with abstract graphs, say, represented by a finite set of edges (type $(\mathfrak{a} \times \mathfrak{a})$ fset), whereas the refinement $\mathsf{traverse}_\mathsf{i}$ represents the graph as a list of edges and the inputs and outputs as lists (we use Haskell-style list comprehension syntax). Using the canonical DDS coiterator dds-of and the refinement relation fset-as-list :: $\mathfrak{a}$ list $\otimes$ $\mathfrak{a}$ fset for implementing finite sets by lists, we get the following refinement theorem. Note that we need the general relator $\mathsf{rel}_\mathsf{dds}$ to lift the refinement relations on the inputs and outputs. (Recall that $\Mapsto$ is the function space relator.)

```
primcorec dds-of :: (𝔰 ⇒ 𝔞 ⇒ 𝔟 × 𝔰) ⇒ 𝔰 ⇒ (𝔞, 𝔟) dds where
  run (dds-of f s) = map× id (dds-of f) ∘ f s
definition traverse :: (𝔞 × 𝔞) fset ⇒ (𝔞 fset, 𝔞 fset) dds where
  traverse G = dds-of (λ𝒱 A. (G[A] − 𝒱, 𝒱 ∪ A)) ∅
definition traverseᵢ :: (𝔞 × 𝔞) list ⇒ (𝔞 list, 𝔞 list) dds where
  traverseᵢ E = dds-of (λ𝒱 A. [y | (x,y) ← E, x ∈ set_list A, y ∉ 𝒱], 𝒱 ∪ set_list A)) ∅
lemma REFINEMENT: (fset-as-list ⤇ rel_dds fset-as-list fset-as-list) traverseᵢ traverse
```

Quotients. Quotient preservation theorems are used to modularly construct quotient types and to lift functions and lemmas to them [16,17,20]. For example, the type of finite sets fset is a quotient of lists where the order and multiplicity of the elements are ignored. Given the quotient preservation theorems for $\Rightarrow$ and dds, Isabelle's Lifting package can lift this fset–list quotient to traverse's type. It can thus synthesise a definition for traverse using $\mathsf{traverse}_\mathsf{i}$ and prove the REFINEMENT lemma automatically given a proof that $\mathsf{traverse}_\mathsf{i}$ respects the quotient.

The refinement relation fset-as-list can additionally be parametrised by a refinement relation R on the elements [20]: fset-as-list$'$ $R = \mathsf{rel}_\mathsf{list}\ R \bigcirc$ fset-as-list. Combining

$\mathsf{traverse_i}$'s parametricity with REFINEMENT using some BNF_{CC} relator properties, one can then automatically derive a stronger refinement rule, where the node type can simultaneously be refined; the assumption expresses that R must preserve the identity of nodes, as expected from $\mathsf{traverse_i}$'s implicit dependence on the equality operation.

$$\frac{(R \mathrel{⇻} R \mathrel{⇻} (=))\ (=)\ (=)}{(\mathsf{fset\text{-}as\text{-}list'}\ R \mathrel{⇻} \mathsf{rel_{dds}}\ (\mathsf{fset\text{-}as\text{-}list'}\ R)\ (\mathsf{fset\text{-}as\text{-}list'}\ R))\ \mathsf{traverse_i}\ \mathsf{traverse}}$$

Generalised rewriting. Rewriting replaces subterms with equal terms. In generalised rewriting, relations other than equality are considered, and the context in which rewriting takes place must have an appropriate congruence property [37]. For example, the DDS seen outputs all the elements in the current input set that it has seen before. It is a monotone system with respect to the subset relation, which we express using the DDS relator. The graph traversal traverse is also monotone in the underlying graph provided that the input sets remain the same.

```
definition seen :: ('a fset, 'a fset) dds where seen = dds-of (λS A. (S ∩ A, S ∪ A)) ∅
lemma SEEN-MONO: rel_dds (⊆) (⊆) seen seen
lemma TRAVERSE-MONO: rel_dds (=) (⊆) (traverse G) (traverse H) if G ⊆ H
```

Now suppose that H is a supergraph of G, or equivalently $G \subseteq H$. Using the parametricity of sequential composition, we can thus rewrite $\mathsf{traverse}\ G \bullet \mathsf{seen}$ to $\mathsf{traverse}\ H \bullet \mathsf{seen}$, where the systems are related by $\mathsf{rel_{dds}}\ (=)\ (\subseteq)$.

2 Bounded Natural Functors with Co- and Contravariance

The operations specified by a BNF act only on live type parameters. As discussed in the introduction, many types admit more general operations, for example the function space's mapper $\mathsf{map}_{⇒}\ g\ h\ f = h \circ f \circ g$. Yet, the BNF structure is restricted to the mapper $\mathsf{map}_{⇒}\ \mathsf{id}\ h$, which targets only the range, but not the domain.

In this section, we define bounded natural functors with covariance and contravariance (BNF_{CC}) as a generalization of BNFs. A BNF_{CC} has a mapper and relator which take additional covariant and contravariant arguments corresponding to (a subset of) the dead parameters $\overline{\mathfrak{d}}$. Thus $\overline{\mathfrak{d}}$ is refined into three disjoint sequences: $\overline{\mathfrak{c}}$ for covariant, $\overline{\mathfrak{k}}$ for contravariant, and $\overline{\mathfrak{f}}$ for the remaining fixed parameters which are ignored by the generalised operations. The names covariant and contravariant indicate whether the mapper preserves the order of composition or swaps it, and whether the relator is monotone or anti-monotone in the corresponding argument, respectively. (Live parameters behave like covariant parameters in this regard. We use "covariant" only for parameters that are not live.) For example, the function space $\mathfrak{k} \Rightarrow \mathfrak{l}$ is a BNF_{CC} that is live in $\mathfrak{l}$ and contravariant in $\mathfrak{k}$, as $\mathsf{map}_{⇒}$'s type $(\mathfrak{k}' \Rightarrow \mathfrak{k}) \Rightarrow (\mathfrak{l} \Rightarrow \mathfrak{l}') \Rightarrow (\mathfrak{k} \Rightarrow \mathfrak{l}) \Rightarrow (\mathfrak{k}' \Rightarrow \mathfrak{l}')$ indicates. Similarly, the BNF_{CC} $(\mathfrak{c} \Rightarrow \mathfrak{k}) \Rightarrow \mathfrak{l}$ is live in $\mathfrak{l}$, covariant in $\mathfrak{c}$, and contravariant in $\mathfrak{k}$.

Definition 1 (BNF_{CC}). *A BNF_{CC} is a type constructor F with operations*

$$\mathit{map}_F :: \overline{(\mathfrak{l} \Rightarrow \mathfrak{l}')} \Rightarrow \overline{(\mathfrak{c} \Rightarrow \mathfrak{c}')} \Rightarrow \overline{(\mathfrak{k}' \Rightarrow \mathfrak{k})} \Rightarrow (\overline{\mathfrak{l}}, \overline{\mathfrak{c}}, \overline{\mathfrak{k}}, \overline{\mathfrak{f}})\ F \Rightarrow (\overline{\mathfrak{l}'}, \overline{\mathfrak{c}'}, \overline{\mathfrak{k}'}, \overline{\mathfrak{f}})\ F$$
$$\mathit{rel}_F :: \overline{\mathfrak{l} \otimes \mathfrak{l}'} \Rightarrow \overline{\mathfrak{c} \otimes \mathfrak{c}'} \Rightarrow \overline{\mathfrak{k} \otimes \mathfrak{k}'} \Rightarrow (\overline{\mathfrak{l}}, \overline{\mathfrak{c}}, \overline{\mathfrak{k}}, \overline{\mathfrak{f}})\ F \otimes (\overline{\mathfrak{l}'}, \overline{\mathfrak{c}'}, \overline{\mathfrak{k}'}, \overline{\mathfrak{f}})\ F$$

and, like for plain BNFs, a cardinality bound bd_F and set functions set^i_F for all live parameters $\mathfrak{l}_i$. The cardinality bound may depend on $\overline{\mathfrak{c}}$, $\overline{\mathfrak{k}}$, and $\overline{\mathfrak{f}}$, but not on $\overline{\mathfrak{l}}$. We define two conditions $\mathsf{pos}_F, \mathsf{neg}_F$ for the relator rel_F subdistributing over relation composition:[2]

$$\mathsf{pos}_F, \mathsf{neg}_F :: \overline{(\mathfrak{c} \otimes \mathfrak{c}') \times (\mathfrak{c}' \otimes \mathfrak{c}'')} \Rightarrow \overline{(\mathfrak{k} \otimes \mathfrak{k}') \times (\mathfrak{k}' \otimes \mathfrak{k}'')} \Rightarrow \mathsf{bool}$$

$$\mathsf{pos}_F\ \overline{(C,C')}\ \overline{(K,K')} \longleftrightarrow \big(\forall \overline{L}\ \overline{L'}.\ \mathsf{rel}_F\ \overline{L}\ \overline{C}\ \overline{K} \circ \mathsf{rel}_F\ \overline{L'}\ \overline{C'}\ \overline{K'} \sqsubseteq \mathsf{rel}_F\ \overline{(L \circ L')}\ \overline{(C \circ C')}\ \overline{(K \circ K')}\big) \tag{5}$$

$$\mathsf{neg}_F\ \overline{(C,C')}\ \overline{(K,K')} \longleftrightarrow \big(\forall \overline{L}\ \overline{L'}.\ \mathsf{rel}_F\ \overline{(L \circ L')}\ \overline{(C \circ C')}\ \overline{(K \circ K')} \sqsubseteq \mathsf{rel}_F\ \overline{L}\ \overline{C}\ \overline{K} \circ \mathsf{rel}_F\ \overline{L'}\ \overline{C'}\ \overline{K'}\big) \tag{6}$$

The BNF_{CC} operations must satisfy the conditions shown in Figure 1:

1. *The mapper map_F is functorial with respect to all non-fixed parameters* (7) *and relationally parametric* (8).
2. *The BNF laws about the setters (the cardinality bound, naturality, and congruence) are satisfied for the mapper $\mathsf{map}^\star_F\ \overline{\ell} = \mathsf{map}_F\ \overline{\ell}\ \overline{id}\ \overline{id}$ restricted to live arguments* (9).
3. *The relator rel_F is monotone in live and covariant arguments, and anti-monotone in contravariant arguments; the relator $\mathsf{rel}^\star_F\ \overline{L} = \mathsf{rel}^\star_F\ \overline{L}\ \overline{(=)}\ \overline{(=)}$ restricted to live arguments is strongly monotone* (10).[3]
4. *The relator preserves equality and distributes over converses $\_^{-1}$* (11).
5. *The relator distributes over relation composition if the relations for covariant and contravariant parameters are equality* (12).

In comparison to plain BNFs, the BNF_{CC} relator is a primitive operation because it is not obvious how to generalise the characterisation (4) in terms of the mapper and setters to covariant and contravariant arguments. We therefore require several properties of the relator. Note that strong monotonicity (10) and negative composition subdistributivity (12) on live arguments are equivalent to the characterisation of $\mathsf{rel}^\star_F$, given the other axioms.

Distributivity over relation composition is split into two directions (positive and negative) because concrete functors satisfy the directions under different conditions and some theorems only need one of the two directions. The names positive and negative stem from Isabelle's Lifting package, which needs the appropriate direction for positive or negative positions in types. In this paper, we often derive sufficient criteria for each direction, for concrete functors and BNF_{CC} constructions. For example, the function space $\mathfrak{k} \Rightarrow \mathfrak{l}$ satisfies the positive direction unconditionally, i.e., $\mathsf{pos}_{\Rightarrow}\ \_ = \mathsf{True}$. In contrast, the negative direction does not always hold. But it does if the contravariant relations are functional, i.e., graphs of functions:

$$\frac{\mathsf{left\text{-}unique}\ K \quad \mathsf{right\text{-}total}\ K \quad \mathsf{right\text{-}unique}\ K' \quad \mathsf{left\text{-}total}\ K'}{\mathsf{neg}_{\Rightarrow}\ (K,K')}, \tag{13}$$

[2] In our formalisation, pos_F and neg_F take type tokens to avoid issues with hidden polymorphism in the live and fixed type parameters. We omit this detail in the paper to simplify the notation.

[3] When $\mathsf{pos}_F\ \overline{((=),C)}\ \overline{((=),K)} = \mathsf{neg}_F\ \overline{((=),C)}\ \overline{((=),K)} = \mathsf{True}$ for all $\overline{C}$ and $\overline{K}$, then the two monotonicity rules (10) are equivalent to the following combined rule:

$$\frac{\forall i.\ \forall a \in \mathsf{set}^i_F\ x.\ \forall b \in \mathsf{set}^i_F\ y.\ L_i\ a\ b \longrightarrow L'_i\ a\ b \qquad \forall i.\ C_i \sqsubseteq C'_i \qquad \forall i.\ K'_i \sqsubseteq K_i}{\mathsf{rel}_F\ \overline{L}\ \overline{C}\ \overline{K}\ x\ y \longrightarrow \mathsf{rel}_F\ \overline{L'}\ \overline{C'}\ \overline{K'}\ x\ y}$$

$$\mathsf{map_F}\ \overline{\mathsf{id}}\ \overline{\mathsf{id}}\ \overline{\mathsf{id}} = \mathsf{id} \qquad \mathsf{map_F}\ (\overline{\ell} \circ \overline{\ell'})\ (\overline{c} \circ \overline{c'})\ (\overline{k'} \circ \overline{k}) = \mathsf{map_F}\ \overline{\ell}\ \overline{c}\ \overline{k} \circ \mathsf{map_F}\ \overline{\ell'}\ \overline{c'}\ \overline{k'} \tag{7}$$

$$\left(\overline{(L \Mapsto L')} \Mapsto \overline{(C \Mapsto C')} \Mapsto \overline{(K' \Mapsto K)} \Mapsto \mathsf{rel_F}\ \overline{L}\ \overline{C}\ \overline{K} \Mapsto \mathsf{rel_F}\ \overline{L'}\ \overline{C'}\ \overline{K'}\right)\ \mathsf{map_F}\ \mathsf{map_F} \tag{8}$$

$$\forall i.\ |\mathsf{set}^i_\mathsf{F}| \leq \mathsf{bd_F} \qquad \forall i.\ \mathsf{set}^i_\mathsf{F}\ (\mathsf{map}^\star_\mathsf{F}\ \overline{\ell}\ x) = \ell_i\ \text{`}\ \mathsf{set}^i_\mathsf{F}\ x \qquad \frac{\forall i.\ \forall y \in \mathsf{set}^i_\mathsf{F}\ x.\ \ell_i\ y = \ell'_i\ y}{\mathsf{map}^\star_\mathsf{F}\ \overline{\ell}\ x = \mathsf{map}^\star_\mathsf{F}\ \overline{\ell'}\ x} \tag{9}$$

$$\frac{\forall i.\ L_i \sqsubseteq L'_i \quad \forall i.\ C_i \sqsubseteq C'_i \quad \forall i.\ K'_i \sqsubseteq K_i}{\mathsf{rel_F}\ \overline{L}\ \overline{C}\ \overline{K} \sqsubseteq \mathsf{rel_F}\ \overline{L'}\ \overline{C'}\ \overline{K'}} \qquad \frac{\forall i.\ \forall a \in \mathsf{set}^i_\mathsf{F}\ x.\ \forall b \in \mathsf{set}^i_\mathsf{F}\ y.\ L_i\ a\ b \longrightarrow L'_i\ a\ b}{\mathsf{rel}^\star_\mathsf{F}\ \overline{L}\ x\ y \longrightarrow \mathsf{rel}^\star_\mathsf{F}\ \overline{L'}\ x\ y} \tag{10}$$

$$\mathsf{rel_F}\ \overline{(=)}\ \overline{(=)}\ \overline{(=)} = (=) \qquad (\mathsf{rel_F}\ \overline{L}\ \overline{C}\ \overline{K})^{-1} = \mathsf{rel_F}\ \overline{L^{-1}}\ \overline{C^{-1}}\ \overline{K^{-1}} \tag{11}$$

$$\mathsf{pos_F}\ \overline{((=),(=))}\ \overline{((=),(=))} \qquad \mathsf{neg_F}\ \overline{((=),(=))}\ \overline{((=),(=))} \tag{12}$$

Fig. 1. Conditions on the operations of a $\mathrm{BNF_{CC}}$

where left-unique $R = (\forall x\ z\ y.\ R\ x\ z \wedge R\ y\ z \longrightarrow x = y)$ and left-total $R = (\forall x.\ \exists y.\ R\ x\ y)$, and right-unique and right-total are defined analogously.

The precise relationship between BNFs and $\mathrm{BNF_{CC}}$s is as follows:

Proposition 1.

1. *Every BNF $(\overline{\mathfrak{l}}, \overline{\mathfrak{d}})$ F is a $\mathrm{BNF_{CC}}$ where $\overline{\mathfrak{l}}$ are live, $\overline{\mathfrak{d}}$ are fixed, and $\mathsf{map_F}$, $\overline{\mathsf{set_F}}$, $\mathsf{bd_F}$, and $\mathsf{rel_F}$ are inherited from the BNF. So $\mathsf{pos_F} = \mathsf{neg_F} = \mathsf{True}$.*
2. *Every $\mathrm{BNF_{CC}}$ $(\overline{\mathfrak{l}}, \overline{\mathfrak{c}}, \overline{\mathfrak{k}}, \overline{\mathfrak{f}})$ F is a BNF with live parameters $\overline{\mathfrak{l}}$ and dead parameters $\overline{\mathfrak{c}}, \overline{\mathfrak{k}}, \overline{\mathfrak{f}}$ for the mapper $\mathsf{map^\star_F}$, setters $\overline{\mathsf{set_F}}$, bound $\mathsf{bd_F}$, and relator $\mathsf{rel^\star_F}$.*

The $\mathrm{BNF_{CC}}$ axioms are either BNF axioms or routinely proved from them, and vice versa. The only exception is $\mathsf{rel^\star_F}$'s equational characterisation (4) for a $\mathrm{BNF_{CC}}$, which implies, e.g., that $\mathsf{neg_F} = \mathsf{True}$ [7]. To show the characterisation, we use the following property, which generalises Lemma 1 to the $\mathrm{BNF_{CC}}$ mapper and relator. It follows from the functor laws (7), parametricity (8), and equality preservation (11).

Lemma 2. *For a $\mathrm{BNF_{CC}}$ F, the graph of $\mathsf{map_F}\ \overline{\ell}\ \overline{c}\ \overline{k}$ is the relator applied to the graphs of $\overline{\ell}$, $\overline{c}$, and the converse graphs of $\overline{k}$: $\mathsf{Gr}\ (\mathsf{map_F}\ \overline{\ell}\ \overline{c}\ \overline{k}) = \mathsf{rel_F}\ \overline{(\mathsf{Gr}\ \ell)}\ \overline{(\mathsf{Gr}\ c)}\ \overline{(\mathsf{Gr}\ k)^{-1}}$.*

We now give some examples of $\mathrm{BNF_{CC}}$s. Every BNF without dead parameters is also a $\mathrm{BNF_{CC}}$ with all parameters being live by Proposition 1. This includes all sums-of-product (co)datatypes, which are also known as polynomial (co)datatypes. Many other BNFs such as distinct lists, finite and countable sets, and discrete probability distributions fall into this class, too. For these, our $\mathrm{BNF_{CC}}$ generalisation would not have been necessary. But there are other types where $\mathrm{BNF_{CC}}$s do make a difference:

(a) We previously mentioned the function type $\mathfrak{k} \Rightarrow \mathfrak{l}$ with mapper $\mathsf{map}_\Rightarrow$ and relator $\Mapsto$, where $\mathfrak{l}$ is live and $\mathfrak{k}$ is contravariant.

(b) The powerset functor $\mathfrak{c}\ \mathsf{set}$ has the image operation as the mapper and the relator

$$\mathsf{rel_{set}}\ C\ X\ Y = (\forall x \in X.\ \exists y \in Y.\ C\ x\ y) \wedge (\forall y \in Y.\ \exists x \in X.\ C\ x\ y).$$

The parameter c is covariant and not live only because there is no bound on the cardinality. We have $\mathsf{pos_{set\ \_}} = \mathsf{neg_{set\ \_}} = \mathsf{True}$.

(c) Sets $\mathfrak{c}$ bset_b with a finite cardinality bound $b \in \mathbb{N}$ are a subtype of the powerset functor $\mathfrak{c}$ set. For $b > 2$, bset_b is not a BNF in $\mathfrak{c}$ [15]. We will see in Sect. 5 that we obtain the $\mathrm{BNF_{CC}}$ properties by composition and subtyping. We have $\mathsf{pos}_{\mathsf{bset}_b\ \_} = \mathsf{True}$, and $\mathsf{right\text{-}unique}\ C \vee \mathsf{left\text{-}unique}\ C'$ implies $\mathsf{neg}_{\mathsf{bset}_b}\ (C, C')$.

(d) Predicates $\mathfrak{k}$ $\mathsf{pred} = \mathfrak{k} \Rightarrow \mathsf{bool}$ are the contravariant powerset functor with mapper $\mathsf{map}_{\Rightarrow}\ k\ \mathsf{id}$ and relator $K \Rightarrow (=)$. Interestingly, the negative subdistributivity condition $\mathsf{neg}_{\mathsf{pred}}$ is weaker than $\mathsf{neg}_{\Rightarrow}$ because the live parameter of $\Rightarrow$ has been instantiated to bool. We thus get that $\mathsf{neg}_{\mathsf{pred}}\ (K, K')$ is implied by $\mathsf{left\text{-}unique}\ K \wedge \mathsf{right\text{-}total}\ K \vee \mathsf{right\text{-}unique}\ K' \wedge \mathsf{left\text{-}total}\ K'$, i.e., only one of the two relations must be functional, not both as in (13). Clearly, $\mathsf{pos}_{\mathsf{pred}\ \_} = \mathsf{True}$.

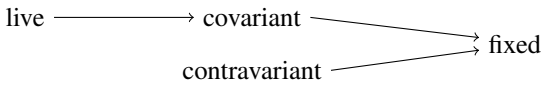
(e) Filters $\mathfrak{c}$ filter (sets of sets closed under finite intersections and supersets) can be viewed as a semantic subtype of $\mathfrak{c}$ $\mathsf{pred}\ \mathsf{pred} = (\mathfrak{c} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{bool}$. Here, $\mathfrak{c}$ is covariant because we go twice through $\Rightarrow$'s left-hand side.

These examples propagate: whenever one of these types occurs inside a larger type, this type also benefits from $\mathrm{BNF_{CC}}$'s greater generality over BNF's.

3 Simple Operations on $\mathrm{BNF_{CC}}$s

We now show that $\mathrm{BNF_{CC}}$s are closed under functor composition, like BNFs are. This property is crucial for a modular construction of (co)datatypes. It allows us to construct arbitrarily complex signatures from simple building blocks, because the $\mathrm{BNF_{CC}}$ properties follow by construction. For example, the type $\mathfrak{k}\ \mathsf{option} \Rightarrow (\mathfrak{c}_1 \times \mathfrak{c}_2)\ \mathsf{set}$ is a composition of the type constructors $\Rightarrow$, option, set, and $\times$. For $\mathrm{BNF_{CC}}$s, we distinguish three kinds of composition depending on whether the composition occurs in a live (set in $\mathfrak{k} \Rightarrow \natural$), covariant ($\times$ in $\natural\ \mathsf{set}$), or contravariant parameter (option in $\natural \Rightarrow \mathfrak{l}$).

Before we turn to composition, we discuss two technical issues: *demoting* and *merging* parameters. For BNFs, demotion is known as killing, which transforms a live parameter into a dead. For $\mathrm{BNF_{CC}}$s, there are three kinds of demotion ($\longrightarrow$):



Demotion is a preparatory step for composition: If composition happens in a covariant or contravariant position, the live parameters of the inner functor are no longer live. Demotion first transforms all live parameters into covariant ones. During composition in a covariant or contravariant parameter, we can thus assume that the inner functor has no live parameters.

Merging unifies two type parameters of a $\mathrm{BNF_{CC}}$. Both type parameters must be of the same kind (live, covariant, contravariant, or fixed)—otherwise, they must be demoted first. For example, we can merge $\mathfrak{c}_1$ and $\mathfrak{c}_2$ in $(\mathfrak{c}_1 \times \mathfrak{c}_2)\ \mathsf{set}$ directly to obtain the unary covariant functor $(\mathfrak{c} \times \mathfrak{c})\ \mathsf{set}$. In contrast, before merging $\mathfrak{k}$ and $\mathfrak{l}$ in $\mathfrak{k} \Rightarrow \mathfrak{l}$, we must demote the live parameter $\mathfrak{l}$ and the contravariant parameter $\mathfrak{k}$ to fixed. Treating merging as a separate operation simplifies the composition theorem (Theorem 1 below) as we can assume without loss of generality that the two functors do not share any parameters.

Proposition 2. *$\mathrm{BNF_{CC}}$s are closed under all kinds of demotion and merging.*

Demoting a live parameter adds an argument to the conditions for composition distribution, i.e., it removes the corresponding relations from the universal quantifiers in (5,6). So the conditions become weaker. It may therefore be useful to associate one type constructor with several $\mathrm{BNF_{CC}}$ instances that differ in the live parameters. In $\mathfrak{k} \Rightarrow \mathfrak{l}$, e.g., demoting $\mathfrak{l}$ to $\mathfrak{c}$ allows us to relax the conditions on $\mathfrak{k}$'s relations by imposing some on $\mathfrak{c}$'s. In the covariant case, negative distributivity $\mathsf{neg}_{\Rightarrow}\ (C,C')\ (K,K')$ holds if right-unique K', left-total K', left-unique C', and right-total C'. But in the live case, $\mathsf{neg}_{\Rightarrow}\ (K,K')$ does not hold for right-unique K', left-total K' in general. This difference will be crucial for quotient preservation (Sect. 6).

We now return to composition and show that the class $\mathrm{BNF_{CC}}$ of functors is closed under composition. We only discuss composition in a single parameter. This is not a restriction because composing with multiple functors simultaneously is equivalent to a sequence of single compositions, independent of the order. We also assume that the two functors do not share any parameters. A subsequent merge step can always introduce the sharing. We distinguish four different kinds of composition depending on which parameter the inner functor instantiates. For each case, we obtain different sufficient criteria for relator subdistributivity, as shown in the next theorem.

Theorem 1. *$\mathrm{BNF_{CC}}$s are closed under composition in all kinds of parameters. Formally, let $(\overline{\mathfrak{l}_F}, \overline{\mathfrak{c}_F}, \overline{\mathfrak{k}_F}, \overline{\mathfrak{f}_F})\ F$ and $(\overline{\mathfrak{l}_G}, \overline{\mathfrak{c}_G}, \overline{\mathfrak{k}_G}, \overline{\mathfrak{f}_G})\ G$ be $\mathrm{BNF_{CC}}$s such that no parameter is shared between F and G. We consider four kinds of composing F with G into a new functor FG, where i denotes the position of the composition in F's corresponding parameter list:*[4]

Live $(\overline{\mathfrak{l}_F^{<i}}, (\overline{\mathfrak{l}_G}, \overline{\mathfrak{c}_G}, \overline{\mathfrak{k}_G}, \overline{\mathfrak{f}_G})\ G, \overline{\mathfrak{l}_F^{\geq i}}, \overline{\mathfrak{c}_F}, \overline{\mathfrak{k}_F}, \overline{\mathfrak{f}_F})\ F$ *is a $\mathrm{BNF_{CC}}$ with $\overline{\mathfrak{l}_F^{\neq i}}, \overline{\mathfrak{l}_G}$ live, $\overline{\mathfrak{c}_F}, \overline{\mathfrak{c}_G}$ covariant, $\overline{\mathfrak{k}_F}, \overline{\mathfrak{k}_G}$ contravariant, and $\overline{\mathfrak{f}_F}, \overline{\mathfrak{f}_G}$ fixed. $\mathsf{pos}_F\ \overline{(C_F,C'_F)}\ \overline{(K_F,K'_F)}$ and $\mathsf{pos}_G\ \overline{(C_G,C'_G)}\ \overline{(K_G,K'_G)}$ are sufficient for $\mathsf{pos}_{FG}\ \overline{(C_F,C'_F)}\ \overline{(C_G,C'_G)}\ \overline{(K_F,K'_F)}\ \overline{(K_G,K'_G)}$; it is the same for neg_{FG}.*

Covariant *If $\overline{\mathfrak{l}_G}$ is empty, then $(\overline{\mathfrak{l}_F}, \overline{\mathfrak{c}_F^{<i}}, (\overline{\mathfrak{c}_G}, \overline{\mathfrak{k}_G}, \overline{\mathfrak{f}_G})\ G, \overline{\mathfrak{c}_F^{\geq i}}, \overline{\mathfrak{k}_F}, \overline{\mathfrak{f}_F})\ F$ is a $\mathrm{BNF_{CC}}$ with $\overline{\mathfrak{l}_F}$ live, $\overline{\mathfrak{c}_F^{\neq i}}, \overline{\mathfrak{c}_G}$ covariant, $\overline{\mathfrak{k}_F}, \overline{\mathfrak{k}_G}$ contravariant, and $\overline{\mathfrak{f}_F}, \overline{\mathfrak{f}_G}$ fixed. $\mathsf{pos}_G\ \overline{(C_G,C'_G)}\ \overline{(K_G,K'_G)}$ and $\mathsf{pos}_F\ \overline{(C_F,C'_F)^{<i}}\ (\mathsf{rel}_G\ \overline{C_G}\ \overline{K_G}, \mathsf{rel}_G\ \overline{C'_G}\ \overline{K'_G})\ \overline{(C_F,C'_F)^{>i}}\ \overline{(K_F,K'_F)}$ are sufficient for $\mathsf{pos}_{FG}\ \overline{(C_F,C'_F)^{\neq i}}\ \overline{(C_G,C'_G)}\ \overline{(K_F,K'_F)}\ \overline{(K_G,K'_G)}$; it is the same for neg_{FG}.*

Contravariant *If $\overline{\mathfrak{l}_G}$ is empty, then $(\overline{\mathfrak{l}_F}, \overline{\mathfrak{c}_F}, \overline{\mathfrak{k}_F^{<i}}, (\overline{\mathfrak{c}_G}, \overline{\mathfrak{k}_G}, \overline{\mathfrak{f}_G})\ G, \overline{\mathfrak{k}_F^{\geq i}}, \overline{\mathfrak{f}_F})\ F$ is a $\mathrm{BNF_{CC}}$ with $\overline{\mathfrak{l}_F}$ live, $\overline{\mathfrak{c}_F}, \overline{\mathfrak{k}_G}$ covariant, $\overline{\mathfrak{k}_F^{\neq i}}, \overline{\mathfrak{c}_G}$ contravariant, and $\overline{\mathfrak{f}_F}, \overline{\mathfrak{f}_G}$ fixed. $\mathsf{neg}_G\ \overline{(C_G,C'_G)}\ \overline{(K_G,K'_G)}$ and $\mathsf{pos}_F\ \overline{(C_F,C'_F)}\ \overline{(K_F,K'_F)^{<i}}\ (\mathsf{rel}_G\ \overline{C_G}\ \overline{K_G}, \mathsf{rel}_G\ \overline{C'_G}\ \overline{K'_G})\ \overline{(K_F,K'_F)^{>i}}$ are sufficient for $\mathsf{pos}_{FG}\ \overline{(C_F,C'_F)}\ \overline{(K_G,K'_G)}\ \overline{(K_F,K'_F)^{\neq i}}\ \overline{(C_G,C'_G)}$; it is the same for neg_{FG}. (Note that in the new functor, $\overline{C_G}, \overline{C'_G}$ are now contravariant and $\overline{K_G}, \overline{K'_G}$ covariant.)*

Fixed *If $\overline{\mathfrak{l}_G}$, $\overline{\mathfrak{c}_G}$, $\overline{\mathfrak{k}_G}$ are all empty, then $(\overline{\mathfrak{l}_F}, \overline{\mathfrak{c}_F}, \overline{\mathfrak{k}_F}, \overline{\mathfrak{f}_F^{<i}}, \overline{\mathfrak{f}_G}\ G, \overline{\mathfrak{f}_F^{\geq i}})\ F$ is a $\mathrm{BNF_{CC}}$ with $\overline{\mathfrak{l}_F}$ live, $\overline{\mathfrak{c}_F}$ covariant, $\overline{\mathfrak{k}_F}$ contravariant, and $\overline{\mathfrak{f}_F^{\neq i}}, \overline{\mathfrak{f}_G}$ fixed. $\mathsf{pos}_F\ \overline{(C_F,C'_F)}\ \overline{(K_F,K'_F)}$ is sufficient for $\mathsf{pos}_{FG}\ \overline{(C_F,C'_F)}\ \overline{(K_F,K'_F)}$; it is the same for neg_{FG}.*

For example, consider the composition of $(\mathfrak{l}_1, \mathfrak{k}_1)\ \mathsf{F} = \mathfrak{k}_1 \Rightarrow \mathfrak{l}_1$ with $(\mathfrak{c}_1, \mathfrak{k}_2)\ \mathsf{G} = \mathfrak{k}_2 \Rightarrow \mathfrak{c}_1$ in the contravariant parameter $\mathfrak{k}_1$. In G, the range $\mathfrak{c}_1$ is normally live, but it has already been demoted such that there are no more live parameters. We obtain the $\mathrm{BNF_{CC}}$

[4] For example, if we instantiate the third covariant parameter of F with G, then $i = 3$.

$(\mathfrak{k}_2 \Rightarrow \mathfrak{c}_1) \Rightarrow \mathfrak{l}_1$, where $\mathfrak{k}_2$ is now covariant and $\mathfrak{c}_1$ is contravariant, while $\mathfrak{l}_1$ remains live. The conditions $\mathsf{pos}_{\Rightarrow}\ (K_2, K_2) = \mathsf{True}$ and $\mathsf{neg}_{\Rightarrow}\ (K_2 \Mapsto C_1, K_2' \Mapsto C_1')$ are sufficient for negative subdistributivity of the composed relator $(K_2 \Mapsto C_1) \Mapsto L_1$, i.e., they imply $\mathsf{neg}_{(\Rightarrow)\Rightarrow}\ (K_2, K_2')\ (C_1, C_1')$.

4 Least and Greatest Fixpoints

Bounded natural functors have been introduced mainly to construct (co)datatypes modularly in HOL. A (co)datatype $\overline{\mathfrak{a}}\ \mathsf{T}$ defined by the command

$$\mathtt{(co)datatype}\ \overline{\mathfrak{a}}\ \mathsf{T} = \mathsf{ctor}_{\mathsf{T}}\ (\mathsf{dtor}_{\mathsf{T}} \colon (\overline{\mathfrak{a}}\ \mathsf{T}, \overline{\mathfrak{a}})\ \mathsf{F})$$

corresponds to the least (greatest) solution X of the fixpoint equation $\overline{\mathfrak{a}}\ X \cong (\overline{\mathfrak{a}}\ X, \overline{\mathfrak{a}})\ \mathsf{F}$, up to the (co)algebra isomorphism given by the constructor $\mathsf{ctor}_{\mathsf{T}}$ and destructor $\mathsf{dtor}_{\mathsf{T}}$. Whenever the (co)recursion goes through a live type parameter of F, the fixpoint exists and it is again a BNF for the remaining live parameters—this is the closure property under fixpoints.[5]

In this section, we show that every (co)datatype defined over a $\mathrm{BNF}_{\mathrm{CC}}$ can be extended to a $\mathrm{BNF}_{\mathrm{CC}}$ in a meaningful way, namely such that the following primitive (co)datatype operations are parametric with respect to the generalised relator: the constructor $\mathsf{ctor}_{\mathsf{T}}$, the destructor $\mathsf{dtor}_{\mathsf{T}}$, and a (co)recursor, which witnesses initiality or finality of the (co)algebra. In the following, we consider a $\mathrm{BNF}_{\mathrm{CC}}$ F and its least fixpoint T taken over the first live parameter. We define T's generalised mapper by primitive (co)recursion according to the fixpoint equation

$$\mathsf{map}_{\mathsf{T}}\ \overline{\ell}\ \overline{c}\ \overline{k}\ (\mathsf{ctor}_{\mathsf{T}}\ x) = \mathsf{ctor}_{\mathsf{T}}\ (\mathsf{map}_{\mathsf{F}}\ (\mathsf{map}_{\mathsf{T}}\ \overline{\ell}\ \overline{c}\ \overline{k})\ \overline{\ell}\ \overline{c}\ \overline{k}\ x),$$

and T's generalised relator (co)inductively as the least or greatest predicate closed under

$$\frac{\mathsf{rel}_{\mathsf{F}}\ (\mathsf{rel}_{\mathsf{T}}\ \overline{L}\ \overline{C}\ \overline{K})\ \overline{L}\ \overline{C}\ \overline{K}\ x\ y}{\mathsf{rel}_{\mathsf{T}}\ \overline{L}\ \overline{C}\ \overline{K}\ (\mathsf{ctor}_{\mathsf{T}}\ x)\ (\mathsf{ctor}_{\mathsf{T}}\ y)}.$$

Note that $\mathsf{rel}_{\mathsf{T}}$ is well-defined since $\mathsf{rel}_{\mathsf{F}}$ is monotone in the live arguments. This choice of the relator (and therefore of the mapper, due to Lemma 2) is intuitively correct as we obtain a general form of parametricity to the extent permitted by $\mathsf{rel}_{\mathsf{F}}$:

Proposition 3. *The constructor, destructor, and (co)recursor for T are parametric with respect to $\mathsf{rel}_{\mathsf{T}}$.*

The canonical BNF map function for T, which acts only on T's live parameters, is equal to $\mathsf{map}_{\mathsf{T}}^{\star}$ by definition. Similarly, the restricted relator $\mathsf{rel}_{\mathsf{T}}^{\star}$ satisfies the BNF characterisation (4). The setters $\overline{\mathsf{set}_{\mathsf{T}}}$ satisfy only the restricted parametricity law $(\mathsf{rel}_{\mathsf{T}}^{\star}\ \overline{L} \Mapsto \mathsf{rel}_{\mathsf{set}}\ L_i)\ \mathsf{set}_{\mathsf{T}}^{i}\ \mathsf{set}_{\mathsf{T}}^{i}$. As they ignore the covariant and contravariant parameters, the general parametricity law $(\mathsf{rel}_{\mathsf{T}}\ \overline{L}\ \overline{C}\ \overline{R} \Mapsto \mathsf{rel}_{\mathsf{set}}\ L_i)\ \mathsf{set}_{\mathsf{T}}^{i}\ \mathsf{set}_{\mathsf{T}}^{i}$ does not make sense and does not hold in general either. For example, the setter for the function space $\mathfrak{k} \Rightarrow \mathfrak{l}$ takes the range of the function. Choosing $K = \bot$, where $\bot$ is the

[5] For mutually recursive (co)datatypes, the solutions are taken over a system of equations instead of a single fixpoint equation. The $\mathrm{BNF}_{\mathrm{CC}}$ theory generalises to systems of equations in the same way as the BNF theory does.

empty relation, and $L = (=)$, then $(K \Rrightarrow L)$ $(\lambda\_.\ \mathsf{True})$ $(\lambda\_.\ \mathsf{False})$, but clearly not $\mathsf{rel_{set}}$ $(=)$ $(\mathsf{range}\ (\lambda\_.\ \mathsf{True}))$ $(\mathsf{range}\ (\lambda\_.\ \mathsf{False}))$.

Theorem 2. *BNF_{CC}s are closed under least and greatest fixpoints through live parameters. In particular, if T is the least or greatest fixpoint through one of F's live parameters, then $pos_F\ \overline{(C,C')}\ \overline{(K,K')}$ implies $pos_T\ \overline{(C,C')}\ \overline{(K,K')}$, and the same for neg_F and neg_T.*

5 Subtypes

In HOL, a new type $\overline{\alpha}\ \mathsf{T}$ is defined by carving out a non-empty subset S of an already existing type $\overline{\alpha}\ \mathsf{F}$. Such a type definition creates an embedding isomorphism $\mathsf{Rep_T} :: \overline{\alpha}\ \mathsf{T} \Rightarrow \overline{\alpha}\ \mathsf{F}$ between $\overline{\alpha}\ \mathsf{T}$ and S with inverse $\mathsf{Abs_T} :: \overline{\alpha}\ \mathsf{F} \Rightarrow \overline{\alpha}\ \mathsf{T}$, where $\mathsf{Abs_T}$ is unspecified outside of S. If F is a BNF, then the new type T can inherit F's BNF structure provided that S is "well-behaved." Biendarra [6] identified the following two conditions on S, from which his Isabelle/HOL command `lift-bnf` derives the BNF properties.

- *Closed under the BNF mapper:* whenever $x \in S$, then $\mathsf{map^{\star}_F}\ \overline{\ell}\ x \in S$; and
- *Reflects projections:* if $\mathsf{map^{\star}_F}\ \overline{\pi_1}\ z \in S$ and $\mathsf{map^{\star}_F}\ \overline{\pi_2}\ z \in S$, then $z \in S$.

Meanwhile, Popescu [32] weakened the second condition as follows: whenever $\mathsf{map^{\star}_F}\ \overline{\pi_1}\ z \in S$ and $\mathsf{map^{\star}_F}\ \overline{\pi_2}\ z \in S$, then there exists $y \in S$ such that $\mathsf{set}^i_\mathsf{F}\ y \subseteq \mathsf{set}^i_\mathsf{F}\ z$ for all i, $\mathsf{map^{\star}_F}\ \overline{\pi_1}\ y = \mathsf{map^{\star}_F}\ \overline{\pi_1}\ z$, and $\mathsf{map^{\star}_F}\ \overline{\pi_2}\ y = \mathsf{map^{\star}_F}\ \overline{\pi_2}\ z$.

In this section, we generalise Biendarra's and Popescu's conditions to $\mathrm{BNF_{CC}}$s:

Theorem 3 ($\mathbf{BNF_{CC}}$ inheritance for subtypes). *Let $(\overline{\mathfrak{l}},\overline{\mathfrak{c}},\overline{\mathfrak{k}},\overline{\mathfrak{f}})\ F$ be a BNF_{CC} and let $(\overline{\mathfrak{l}},\overline{\mathfrak{c}},\overline{\mathfrak{k}},\overline{\mathfrak{f}})\ T$ be isomorphic to the non-empty set $S :: (\overline{\mathfrak{l}},\overline{\mathfrak{c}},\overline{\mathfrak{k}},\overline{\mathfrak{f}})\ F\ set$ via the morphisms Rep_T and Abs_T. The type T inherits the BNF_{CC} structure from F via*

$$map_T\ \overline{\ell}\ \overline{c}\ \overline{k} = Abs_T \circ map_F\ \overline{\ell}\ \overline{c}\ \overline{k} \circ Rep_T \qquad set^i_T = set^i_F \circ Rep_T \qquad bd_T = bd_F$$
$$rel_T\ \overline{L}\ \overline{C}\ \overline{K}\ x\ y = rel_F\ \overline{L}\ \overline{C}\ \overline{K}\ (Rep_T\ x)\ (Rep_T\ y)$$

if $rel_T\ \overline{(L \bigcirc L')}\ \overline{(=)}\ \overline{(=)} \sqsubseteq rel_T\ \overline{L}\ \overline{(=)}\ \overline{(=)} \bigcirc rel_T\ \overline{L'}\ \overline{(=)}\ \overline{(=)}$ for all $\overline{L},\overline{L'}$, and $x \in S$ implies $map_F\ \overline{\ell}\ \overline{c}\ \overline{k}\ x \in S$. Moreover, $pos_F\ \overline{(C,C')}\ \overline{(K,K')}$ implies $pos_T\ \overline{(C,C')}\ \overline{(K,K')}$.

Negative subdistributivity can often be reduced to proving closedness under zippings, which generalises reflection of projections in the BNF case. We allow a condition $\mathsf{neg'_T}$ that is stronger than $\mathsf{neg_F}$, assuming that $\mathsf{neg'_T}\ \overline{((=),(=))}\ \overline{((=),(=))}$ still holds. The set S is *closed under zippings for $\mathsf{neg'_T}$* iff

$$\frac{x \in S \qquad y \in S \qquad \mathsf{rel_F}\ \overline{L}\ \overline{(C \bigcirc C')}\ \overline{(K \bigcirc K')}\ x\ y \qquad \mathsf{rel_F}\ \overline{(\lambda a\ (a',b).\ a' = a \land L\ a\ b)}\ \overline{C}\ \overline{K}\ x\ z \qquad \mathsf{rel_F}\ \overline{(\lambda (a,b')\ b.\ b' = b \land L\ a\ b)}\ \overline{C'}\ \overline{K'}\ z\ y}{z \in S}$$

for all x, y, z and all $\overline{L}, \overline{C}, \overline{C'}, \overline{K}, \overline{K'}$ such that $\mathsf{neg'_T}\ \overline{(C,C')}\ \overline{(K,K')}$.

Lemma 3. *Let S be closed under zippings for neg'_T. Then $neg'_T\ \overline{(C,C')}\ \overline{(K,K')}$ implies $neg_T\ \overline{(C,C')}\ \overline{(K,K')}$.*

Corollary 1. *BNF_{CC}s are closed under subtypes that are closed under the BNF_{CC} mapper and zippings (for some condition on negative subdistributivity).*

Non-uniform (co)datatypes are therefore also $\mathrm{BNF_{CC}}$s, as they are defined as subtypes of ordinary (co)datatypes [8], and the subtype predicate is invariant under the mapper.

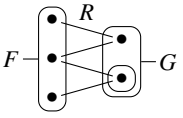
The assumptions on S in Theorem 3 and Corollary 1 are indeed generalisations of Popescu's and Biendarra's conditions, respectively. For when there are neither covariant nor contravariant parameters, the assumptions on S in Theorem 3 are equivalent to Popescu's conditions, given the BNF relator characterisation (4). Similarly, closure under zippings is equivalent to Biendarra's reflecting projections in that case.

Note that closure under zippings strictly implies negative subdistributivity. For example, sets of cardinality at most two are a BNF and a subtype of the finite powerset BNF fset. Yet, the cardinality restriction to at most two does not reflect projections (take $z = \{a,b\} \times \{0,1\}$). Our Theorem 3 handles this case, but Lemma 3 cannot be used as closedness under zippings is not provable. The current implementation of `lift-bnf` cannot handle this case either.

Since $\mathrm{BNF_{CC}}$s do not require the relator distributing unconditionally over relation composition, there can be several relators that extend the mapper in the sense of Lemma 2. For example, filters (Sect. 2) are a subtype of the $\mathrm{BNF_{CC}}$ obtained by composing the contravariant powerset functor with itself. This view yields the mapper $\mathsf{map}_{\mathsf{filter}}\ c\ F = \{X \mid c^{-1}(X) \in F\}$ from the literature. (We omit the conversions between sets and predicates for clarity). Yet, there are two relator canditates for filter: First, the construction in Theorem 3 gives $\mathsf{rel}^1_{\mathsf{filter}}\ R\ F\ G = \mathsf{rel}_{\mathsf{pred}}\ (\mathsf{rel}_{\mathsf{pred}}\ R)\ F\ G$. Second, the canonical categorical extension of a functor on SET to REL [12,34] gives

$$\mathsf{rel}^2_{\mathsf{filter}}\ R\ F\ G = (\exists Z.\ R \in Z \wedge F = \{U \mid \pi_1^{-1}(U) \cap R \in Z\} \wedge G = \{V \mid \pi_1^{-1}(V) \cap R \in Z\})$$

where $f^{-1}(V)$ denotes the preimage of V under f. The latter relator is strictly stronger than the former. For example, the drawing on the right shows a filter $F = \{\{a_1,a_2,a_3\}\}$ on a three-element type, a filter $G = \{\{b_1\},\{b_1,b_2\}\}$ on a two-element type, and a relation R between the elements. We have $\mathsf{rel}^1_{\mathsf{filter}}\ R\ F\ G$, but not $\mathsf{rel}^2_{\mathsf{filter}}\ R\ F\ G$. In this case, $\mathsf{rel}^2_{\mathsf{filter}}$ is the right choice as it gives $\mathsf{pos}_{\mathsf{filter}\ \_} = \mathsf{neg}_{\mathsf{filter}\ \_} = \mathsf{True}$ [12]. But sometimes the relator definition from Theorem 3 is better. Probability distributions with a finite cardinality bound on the support, e.g., preserve quotients only with the relator from Theorem 3 (Sect. 6).



6 Quotient Preservation

We now consider quotient relationships between types and how $\mathrm{BNF_{CC}}$s preserve such relationships. This allows a modular construction of quotients by composing $\mathrm{BNF_{CC}}$s.

A type $\mathfrak{a}$ is a *quotient* of another type $\mathfrak{r}$ under a partial equivalence relation R on $\mathfrak{r}$ iff $\mathfrak{a}$ is isomorphic to $\mathfrak{r}$'s equivalence classes. A quotient $\mathfrak{a}$ can thus be viewed as an abstraction of $\mathfrak{r}$, and, conversely, $\mathfrak{r}$ as a refinement of $\mathfrak{a}$. (This definition subsumes both subtypes and total quotients. One must consider partial equivalence relations in a higher-order setting for reasons similar to why parametricity uses relations instead of functions [16].) A type constructor $\overline{\mathfrak{b}}\ \mathsf{F}$ *preserves quotients* in the type parameters $\overline{\mathfrak{b}^{\in I}} = \mathfrak{b}_{i_1},\ldots,\mathfrak{b}_{i_m}$ iff $\overline{\mathfrak{a}}\ \mathsf{F}$ is a quotient of $\overline{\mathfrak{r}}\ \mathsf{F}$ whenever $\overline{\mathfrak{a}^{\in I}}$ are quotients of $\overline{\mathfrak{r}^{\in I}}$ and $\overline{\mathfrak{a}^{\notin I}} = \overline{\mathfrak{r}^{\notin I}}$ (for some construction of

the equivalence relation; we provide the details below). For lists, e.g., a quotient between element types $\mathfrak{a}$ and $\mathfrak{r}$ yields a quotient between lists of such elements, $\mathfrak{a}$ list and $\mathfrak{r}$ list. Note that quotient preservation is different from the construction of a quotient type or subtype from a $\mathrm{BNF_{CC}}$. The former, which we discuss in this section, deals with type instantiation, while the latter produces a truly new type.

In HOL, a quotient between types is described by a relation $Q :: \mathfrak{r} \otimes \mathfrak{a}$ that is right-total and right-unique. Such a relation induces (i) an embedding morphism *rep* $:: \mathfrak{a} \Rightarrow \mathfrak{r}$, (ii) an abstraction morphism *abs* $:: \mathfrak{r} \Rightarrow \mathfrak{a}$, and (iii) the underlying partial equivalence relation $R = Q \circ Q^{-1}$. The embedding *rep* picks an unspecified element in the equivalence class, which may require the axiom of choice, and *abs r* is unspecified if no equivalence class contains *r*. Due to this underspecification, it is useful to keep track of *rep* and *abs* as primitive operations, e.g., for code generation. Similarly, Isabelle's Lifting package [17] maintains the explicit characterisation of the equivalence relation *R* to simplify the respectfulness proof obligations presented to the user. The predicate Quot formalises these relationships:

$$\mathsf{Quot}\ R\ \mathit{abs}\ \mathit{rep}\ Q \longleftrightarrow (Q \leq \mathsf{Gr}\ \mathit{abs} \wedge \mathsf{Gr}\ \mathit{rep} \leq Q^{-1} \wedge R = Q \circ Q^{-1}). \tag{14}$$

Quotient preservation can thus be expressed as an implication. For lists, e.g., we have that Quot R *abs rep* Q implies Quot $(\mathsf{rel}_{\mathsf{list}}\ R)$ $(\mathsf{map}_{\mathsf{list}}\ \mathit{abs})$ $(\mathsf{map}_{\mathsf{list}}\ \mathit{rep})$ $(\mathsf{rel}_{\mathsf{list}}\ Q)$. Note how the relator and mapper lift the relations and morphisms from elements to lists. BNFs preserve quotients in all live parameters; this is an easy consequence of relator monotonicity and distributivity.

Theorem 4 ([20, Sect. 4.7]). *BNFs preserve quotients in live parameters, in the following sense:* Quot $(\mathit{rel}_F\ \overline{R})$ $(\mathit{map}_F\ \overline{\mathit{abs}})$ $(\mathit{map}_F\ \overline{\mathit{rep}})$ $(\mathit{rel}_F\ \overline{Q})$ *holds whenever* $(\bar{\mathfrak{l}}, \bar{\mathfrak{d}})$ F *is a BNF and* $\forall i$. Quot R_i abs_i rep_i Q_i.

This theorem does not fully generalise to $\mathrm{BNF_{CC}}$s with covariant and contravariant parameters, as the counterexample in Appendix B (Online Resource) shows. We obtain the following result, however, which shows that positive subdistributivity of the relator over the quotient relations and their converses is a sufficient condition for quotient preservation.

Theorem 5. *Let* $(\bar{\mathfrak{l}}, \bar{\mathfrak{c}}, \bar{\mathfrak{k}}, \bar{\mathfrak{f}})$ F *be a* $\mathrm{BNF_{CC}}$. *Assume that* $\forall i$. Quot R^i_χ abs^i_χ rep^i_χ T^i_χ *for all* $\chi \in \{\mathrm{L}, \mathrm{C}, \mathrm{K}\}$. *If* $\mathit{pos}_F\ \overline{(Q_\mathrm{C}, Q_\mathrm{C}^{-1})}\ \overline{(Q_\mathrm{K}, Q_\mathrm{K}^{-1})}$, *then*

$$\mathit{Quot}\ (\mathit{rel}_F\ \overline{R_\mathrm{L}}\ \overline{R_\mathrm{C}}\ \overline{R_\mathrm{K}})\ (\mathit{map}_F\ \overline{\mathit{abs}_\mathrm{L}}\ \overline{\mathit{abs}_\mathrm{C}}\ \overline{\mathit{rep}_\mathrm{K}})\ (\mathit{map}_F\ \overline{\mathit{rep}_\mathrm{L}}\ \overline{\mathit{rep}_\mathrm{C}}\ \overline{\mathit{abs}_\mathrm{K}})\ (\mathit{rel}_F\ \overline{Q_\mathrm{L}}\ \overline{Q_\mathrm{C}}\ \overline{Q_\mathrm{K}}).$$

We now illustrate how this theorem applies to different $\mathrm{BNF_{CC}}$s. Note that it applies to all the $\mathrm{BNF_{CC}}$s mentioned at the end of Sect. 2, as their relators all positively distribute over all relation compositions (if we use the right relator for filters as dicussed in Sect. 5). For a $\mathrm{BNF_{CC}}$ F constructed from these primitives, $\mathsf{pos}_\mathsf{F}\ \_ = \mathsf{True}$ need not hold, though, as $\mathrm{BNF_{CC}}$ composition in negative positions swaps the positive and negative conditions. Nevertheless, we can derive $\mathsf{pos}_\mathsf{F}\ (Q, Q^{-1})$ for quotient relations Q by using our composition theorems, as the following two examples illustrate. First, predicates over predicates $\mathfrak{c}\ \mathsf{pp} = (\mathfrak{c} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{bool}$ do preserve quotients. By the contravariant case of Theorem 1, $\mathsf{pos}_\mathsf{pp}\ (Q, Q^{-1})$ follows from $\mathsf{pos}_\mathsf{pred}\ (Q \Rightarrowtriangle (=), Q^{-1} \Rightarrowtriangle (=))$ and

$\mathsf{neg}_{\mathsf{pred}}\ (Q, Q^{-1})$. The former is trivial as $\mathsf{pos}_{\mathsf{pred}}\ \_ = \mathsf{True}$. For the latter, observe that predicates $\mathfrak{k}$ pred are obtained from the function space $\mathfrak{k} \Rightarrow \mathfrak{c}$ by instantiating $\mathfrak{c}$ with the nullary $\mathrm{BNF_{CC}}$ bool. So, by Theorem 1 (the covariant case), $\mathsf{neg}_{\mathsf{pred}}\ (Q, Q^{-1})$ follows from $\mathsf{neg}_{\mathsf{bool}} = \mathsf{True}$ and $\mathsf{neg}_{\Rightarrow}\ ((=),(=))\ (Q, Q^{-1})$, which is easily proved using Q being a quotient relation. In this reasoning, it is essential that we do not use the function space $\mathrm{BNF_{CC}}$ with the live codomain. Instead, we first demote the codomain to a covariant parameter (fixed would also do). For in the live case, Theorem 1 gives us only the implication from $\mathsf{neg}_{\Rightarrow}\ (Q, Q^{-1})$ (without the live parameter relations as arguments) to $\mathsf{neg}_{\mathsf{pred}}\ (Q, Q^{-1})$, but $\mathsf{neg}_{\Rightarrow}\ (Q, Q^{-1})$ does not hold as it quantifies over all live parameter relations. This illustrates the weakening by demotion that we discussed below Proposition 2.

The second example shows that it is important to associate several $\mathrm{BNF_{CC}}$s with one type constructor, even in a single type expression. The codatatype

$$\mathsf{codatatype}\ (\mathfrak{c}, \mathfrak{k})\ \mathsf{T} = \mathsf{ctor}_{\mathsf{T}}\ ((\mathfrak{c} \Rightarrow \mathfrak{k}) \Rightarrow (\mathfrak{c}, \mathfrak{k})\ \mathsf{T})$$

is the final coalgebra of the functor $(\mathfrak{l}, \mathfrak{c}, \mathfrak{k})\ \mathsf{F} = (\mathfrak{c} \Rightarrow \mathfrak{k}) \Rightarrow \mathfrak{l}$ and it preserves quotients. To derive $\mathsf{pos}_{\mathsf{T}}\ (C, C')\ (K, K')$ modularly from the construction, we must treat F's outer function space with live codomain (as the corecursion goes through this parameter) and $\mathsf{F}'s$ inner function space with covariant codomain (for the same reason as in the pp case).

7 Related Work

We have already discussed the related work on bounded natural functors [6,7,8,20,38] in the previous sections. Here, we discuss how $\mathrm{BNF_{CC}}$s fit into the Isabelle ecosystem, and compare our approach to previous work for other theorem provers.

The Transfer package by Huffman and Kunčar [17] implements Mitchell's representation independence [27] using a database of parametricity theorems and (conditional) respectfulness theorems for equality and quantifiers. $\mathrm{BNF_{CC}}$ relators can be directly used in the parametricity rules, making them more versatile than BNF relators thanks to the generalisation to covariant and contravariant arguments. The respectfulness theorems follow from monotonicity and positive or negative relator distributivity, whose preconditions our composition theorems carefully track. Moreover, Gilcher's automatic derivation of parametricity theorems [11] also benefits from the generalised relators.

The Lifting package [17] lifts constants over quotients and derives appropriate transfer rules using databases of quotient preservation theorems and relator monotonicity and distributivity. Like for Transfer, our theorems can be fed directly into these databases, making the Lifting package more useful.

Lammich's Autoref tool [21,22] performs data refinement based on parametricity. Currently, Lammich must manually derive relators for (co)datatypes. $\mathrm{BNF_{CC}}$s offer a systematic way to define relators and to derive their fundamental properties.

Apart from HOL, parametricity has recently received a lot of attention in dependent type theories as implemented in Coq, Agda, and Lean. In these rich logics, it is possible to internalise Reynolds' relational interpretation of types [5]. So, the parametricity theorem is just a syntactic translation of a type and its proof can be systematically programmed. Various such translations have been studied for different subsets of the logics [3,19,2];

Anand and Morrisett provide a good overview [2]. These works prove (by induction over the syntax of the logic) that all functions defineable in the logic are parametric and then implement this proof as a tool such as ParamCoq [19] and ParamCoq-iff [2]. As HOL lacks the syntactic nature of type theories and its classical axioms forbid a general parametricity result, we follow a semantic approach using BNF_{CC}s instead. This has the advantage that our approach is modular: only semantic properties matter, but not the particular way that something was defined in.

Moreover, most of the syntax-directed type-theoretic works hardly study how the relational interpretation can be used. At best, free theorems are derived (e.g., Anand and Morrisett derive respectfulness of α-equivalence of λ-terms from an operational semantics being parametric). Parametricity is also the foundation for two data refinement frameworks in Coq, Fiat [10] and CoqEAL [9], similar to Autoref [22] in Isabelle/HOL. They define the relators manually in an ad hoc way and it is unclear whether the syntax-directed works could be used instead. In contrast, BNF_{CC}s provide a framework to systematically define mappers and relators and to derive their rich properties. They thus directly lead to a wealth of applications, including free theorems, data refinement, and type abstraction through quotients.

8 Conclusion and Future Work

BNF_{CC}s generalise the concept of bounded natural functors, which are motivated by the construction of (co)datatypes in HOL. They equip both covariant and contravariant type parameters with a functorial structure, even when they do not meet the requirements of bounded naturality. Hence, the mapper and relator of a BNF_{CC} act on these type parameters, too. We have shown that BNF_{CC}s are closed under the most important type construction mechanisms in HOL: composition, datatypes, codatatypes, and subtypes. This way, we obtain canonical definitions of the mapper and the relator for these constructions, together with proofs of some useful properties. For (co)datatypes, it is crucial that we stay compatible with the BNF restrictions, which motivates our unified view on the functorial structure of types. Applications of parametricity, such as data refinement, quotients, and generalised rewriting, benefit from the extended operations.

We have not yet automated the BNF_{CC} construction in Isabelle/HOL, but we have formalised the constructions and proofs in an abstract setting. Moreover, we applied the BNF_{CC} theory manually in a few applications. In the CryptHOL framework [4,24], e.g., the first author manually defined the generalised mapper and relator for the codatatype

$$\mathsf{codatatype}\ (\mathfrak{a},\mathfrak{b},\mathfrak{c})\ \mathsf{gpv} = \mathsf{GPV}\ ((\mathfrak{a} + (\mathfrak{b} \times (\mathfrak{c} \Rightarrow (\mathfrak{a},\mathfrak{b},\mathfrak{c})\ \mathsf{gpv})))\ \mathsf{option}\ \mathsf{pmf})$$

which models sub-probabilistic discrete systems, and proved properties like relator monotonicity and distributivity. Following the BNF_{CC} theory, we have refactored the definitions and proofs. By exploiting the modularity, they became cleaner, simpler, and shorter.

BNF_{CC}s are functors on the category of sets, but for covariant and contravariant parameters, they need not be functors on the category of relations, as the relator need not distribute unconditionally over relation composition [17]. This is a necessary consequence of dealing with the full function space. Therefore, the relator is not uniquely determined by the mapper, either, and one must choose the relator that fits one's needs best.

There are now four groups of type parameters: live, covariant, contravariant, and fixed. Are they enough or do we need further refinements? In the category of sets, this is as far as we can possibly get while retaining the functorial structure. But in some cases, we would like to go beyond. For example, the state $\mathfrak{s}$ in a state monad $(\mathfrak{s}, \mathfrak{a})$ $\mathsf{stateM} = \mathfrak{s} \Rightarrow \mathfrak{a} \times \mathfrak{s}$ occurs in a positive and a negative position, so demotion makes $\mathfrak{s}$ fixed. The $\mathrm{BNF}_{\mathrm{CC}}$ mapper and relator therefore ignore it. One could generalise the mapper to $\mathfrak{s}$ if we restrict the morphisms to bijections, i.e., change the underlying category to bijections. Similarly, if a type parameter has a type class constraint, only type class homomorphisms can be mapped in general. Extending $\mathrm{BNF}_{\mathrm{CC}}$s into this direction is left as future work.

Moreover, we have not studied whether quotient types [17,18] can be equipped with a $\mathrm{BNF}_{\mathrm{CC}}$ structure in general. We are still working on identifying the conditions under which a quotient inherits the BNF structure from the raw type. For the extension to $\mathrm{BNF}_{\mathrm{CC}}$s, we conjecture that we must first generalise the setter concept from live to covariant and contravariant parameters, as unsound (set) functors seem to require repair even in the BNF case [1]. Furthermore, we are interested in lifting a family of quotient relations between two $\mathrm{BNF}_{\mathrm{CC}}$s to a quotient relation between their fixpoints. This is necessary for refining a whole collection of types that is closed under (co)datatype formation, as needed, e.g., in [35].

Acknowledgements. The authors thank Dmitriy Traytel, Andrei Popescu, and the anonymous reviewers for inspiring discussions and suggestions how to improve the presentation. The authors are listed alphabetically.

References

1. Adámek, J., Gumm, H.P., Trnková, V.: Presentation of set functors: A coalgebraic perspective. J. Log. Comput. 20, 991–1015 (2010)
2. Anand, A., Morrisett, G.: Revisiting parametricity: Inductives and uniformity of propositions. CoRR abs/1705.01163 (2017), http://arxiv.org/abs/1705.01163
3. Atkey, R., Ghani, N., Johann, P.: A relationally parametric model of dependent type theory. In: POPL 2014. pp. 503–515. ACM (2014)
4. Basin, D., Lochbihler, A., Sefidgar, S.R.: CryptHOL: Game-based proofs in higher-order logic. Cryptology ePrint Archive: Report 2017/753, https://eprint.iacr.org/2017/753 (2017)
5. Bernardy, J.P., Jansson, P., Paterson, R.: Proofs for free: Parametricity for dependent types. Journal of Functional Programming 22(2), 107–152 (2012)
6. Biendarra, J.: Functor-preserving type definitions in Isabelle/HOL. Bachelor thesis, Fakultät für Informatik, Technische Universität München (2015)
7. Blanchette, J.C., Hölzl, J., Lochbihler, A., Panny, L., Popescu, A., Traytel, D.: Truly modular (co)datatypes for Isabelle/HOL. In: ITP 2014. LNCS, vol. 8558, pp. 93–110. Springer (2014)
8. Blanchette, J.C., Meier, F., Popescu, A., Traytel, D.: Foundational nonuniform (co)datatypes for higher-order logic. In: LICS 2017. pp. 1–12. IEEE (2017)
9. Cohen, C., Dénès, M., Mörtberg, A.: Refinements for free! In: CPP 2013. LNCS, vol. 8307, pp. 147–162. Springer (2013)
10. Delaware, B., Pit-Claudel, C., Gross, J., Chlipala, A.: Fiat: Deductive synthesis of abstract data types in a proof assistant. In: POPL 2015. pp. 689–700. ACM (2015)

11. Gilcher, J., Lochbihler, A., Traytel, D.: Conditional parametricity in Isabelle/HOL (extended abstract). Poster at TABLEAU/FroCoS/ITP 2017, http://www.andreas-lochbihler.de/pub/gilcher2017ITP.pdf (2017)
12. Gumm, H.P.: Functors for coalgebras. Algebra univers. 45, 135–147 (2001)
13. Gunter, E.L.: Why we can't have SML-style datatype declarations in HOL. In: TPHOLs 1992. IFIP Transactions, vol. A-20, pp. 561–568. North-Holland/Elsevier (1992)
14. Haftmann, F., Krauss, A., Kunčar, O., Nipkow, T.: Data refinement in Isabelle/HOL. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 100–115. Springer (2013)
15. Hölzl, J., Lochbihler, A., Traytel, D.: A formalized hierarchy of probabilistic system types. In: Urban, C., Zhang, X. (eds.) ITP 2015. Lecture Notes in Computer Science, vol. 9236, pp. 203–220. Springer (2015)
16. Homeier, P.V.: A design structure for higher order quotients. In: TPHOLs 2005. LNCS, vol. 3603, pp. 130–146. Springer (2005)
17. Huffman, B., Kuncar, O.: Lifting and Transfer: A modular design for quotients in Isabelle/HOL. In: CPP 2013. LNCS, vol. 8307, pp. 131–146. Springer (2013)
18. Kaliszyk, C., Urban, C.: Quotients revisited for Isabelle/HOL. In: SAC 2011. pp. 1639–1644. ACM (2011)
19. Keller, C., Lasson, M.: Parametricity in an impredicative sort. CoRR abs/1209.6336 (2012), http://arxiv.org/abs/1209.6336
20. Kunčar, O.: Types, abstraction and parametric polymorphism in higher-order logic. Ph.D. thesis, Fakultät für Informatik, Technische Universität München (2016)
21. Lammich, P.: Automatic data refinement. In: ITP 2013. LNCS, vol. 7998, pp. 84–99. Springer (2013)
22. Lammich, P., Lochbihler, A.: Automatic refiement to efficient data structures: A comparison of two approaches. Journal of Automated Reasoning (2018)
23. Leino, K.R.M.: Dafny: An automatic program verifier for functional correctness. In: LPAR 2010. LNCS, vol. 6355, pp. 348–370. Springer (2010)
24. Lochbihler, A.: CryptHOL. Archive of Formal Proofs (2017), http://isa-afp.org/entries/CryptHOL.html, Formal proof development
25. Lochbihler, A., Schneider, J.: Bounded natural functors with covariance and contravariance. Archive of Formal Proofs (2018), http://isa-afp.org/entries/BNF_CC.html, Formal proof development
26. Maurer, U.: Indistinguishability of random systems. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 110–132. Springer (2002)
27. Mitchell, J.C.: Representation independence and data abstraction. In: POPL 1986. pp. 263–276. ACM (1986)
28. de Moura, L.M., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The Lean theorem prover (system description). In: CADE 2015. LNCS, vol. 9195, pp. 378–388. Springer (2015)
29. Norell, U.: Towards a practical programming language based on dependent type theory. Ph.D. thesis, Department of Computer Science and Engineering, Chalmers University of Technology (2007)
30. Owre, S., Shankar, N.: Abstract datatypes in PVS. Tech. Rep. CSL-93-9R, Computer Science Laboratory, SRI International (1993)
31. Paulin-Mohring, C.: Inductive definitions in the system Coq – rules and properties. In: TLCA 1993. LNCS, vol. 664, pp. 328–345. Springer (1993)
32. Popescu, A.: Personal communication (2017)
33. Reynolds, J.C.: Types, abstraction and parametric polymorphism. In: IFIP 1983. Information Processing, vol. 83, pp. 513–523. North-Holland/IFIP (1983)
34. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theor. Comput. Sci. 249(1), 3–80 (2000)

35. Schneider, J.: Formalising the run-time costs of HOL programs. Master's thesis, Department of Computer Science, ETH Zurich (2017)
36. Slind, K., Norrish, M.: A brief overview of HOL4. In: TPHOLs 2008. LNCS, vol. 5170, pp. 28–32. Springer (2008)
37. Sozeau, M.: A new look at generalized rewriting in type theory. J. Formalized Reasoning 2(1), 41–62 (2009)
38. Traytel, D., Popescu, A., Blanchette, J.C.: Foundational, compositional (co)datatypes for higher-order logic. In: LICS 2012. pp. 596–605. IEEE (2012)
39. Wadler, P.: Theorems for free! In: FPCA 1989. pp. 347–359. ACM (1989)