Logical Methods in Computer Science
Volume 16, Issue 1, 2020, pp. 8:1–8:63
https://lmcs.episciences.org/

Submitted Jun. 15, 2018
Published Jan. 31, 2020

EFFICIENT AND MODULAR COALGEBRAIC PARTITION REFINEMENT

THORSTEN WISSMANN, ULRICH DORSCH, STEFAN MILIUS, AND LUTZ SCHRÖDER

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
e-mail address: {thorsten.wissmann,ulrich.dorsch,stefan.milius,lutz.schroeder}@fau.de

Abstract. We present a generic partition refinement algorithm that quotients coalgebraic systems by behavioural equivalence, an important task in system analysis and verification. Coalgebraic generality allows us to cover not only classical relational systems but also, e.g. various forms of weighted systems and furthermore to flexibly combine existing system types. Under assumptions on the type functor that allow representing its finite coalgebras in terms of nodes and edges, our algorithm runs in time $\mathcal{O}(m \cdot \log n)$ where n and m are the numbers of nodes and edges, respectively. The generic complexity result and the possibility of combining system types yields a toolbox for efficient partition refinement algorithms. Instances of our generic algorithm match the run-time of the best known algorithms for unlabelled transition systems, Markov chains, deterministic automata (with fixed alphabets), Segala systems, and for color refinement.

1. Introduction

The minimization of a state based system typically consists of two steps:

(1) Removal of unreachable states.

(2) Identification of states exhibiting the same behaviour, w.r.t. a suitable notion of 'sameness'; here we are interested in *minimization under bisimilarity*.

The computation of reachable states is usually accomplished by a straightforward search through the transition graph of a system. Minimization under bisimilarity however is more complex because of its corecursive nature: whether two states are bisimilar depends on which of their successors are bisimilar. In the present work, we present a generic algorithm to perform bisimilarity minimization efficiently for a broad class of systems.

The task of minimization appears as a subtask in state space reduction (e.g. [BO05]) or non-interference checking [vdMZ07]. The notion of bisimulation was first defined for relational systems [vB77, Mil80, Par81]; it was later extended to other system types including probabilistic systems [LS91, DEP02], weighted automata [Buc08], and (weighted) tree automata [HMM09, HMM07]. More generally, *universal coalgebra* (see e.g. Rutten [Rut00]) provides a framework capturing all these types of systems uniformly, and their notions of

Work performed as part of the DFG-funded project COAX (MI 717/5-1 and SCHR 1118/12-1) .

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.23638/LMCS-16(1:8)2020
© T. Wißmann, U. Dorsch, S. Milius, and L. Schröder
Creative Commons

bisimulation appear as special instances of Aczel and Mendler's notion of bisimulation for coalgebras [AM89].

The importance of minimization under bisimilarity appears to increase with the complexity of the underlying system type. E.g. while in LTL model checking, minimization drastically reduces the state space but, depending on the application, does not necessarily lead to a speedup in the overall balance [FV02], in probabilistic model checking, minimization under strong bisimilarity does lead to substantial efficiency gains [KKZJ07]. This is the reason why model checkers implement bisimilarity minimization, e.g. the mCRL2 toolset [BGK$^+$19] provides explicit routines for comparing and minimizing systems w.r.t. strong bisimilarity and also other types of equivalences.

The algorithmics of minimization, often referred to as *partition refinement* or *lumping*, has received a fair amount of attention. Since bisimilarity is a greatest fixpoint, it is more or less immediate that it can be calculated in polynomial time by approximating this fixpoint from above following Kleene's fixpoint theorem. For transition systems, Kanellakis and Smolka [KS83, KS90] introduced an algorithm that in fact runs in time $\mathcal{O}(nm)$ where n is the number of nodes and m is the number of transitions. An even more efficient algorithm running in time $\mathcal{O}(m \log n)$ was later described by Paige and Tarjan [PT87]; this bound holds even if the number of action labels is not fixed [Val09]. Current algorithms typically apply further optimizations to the Paige-Tarjan algorithm, achieving better average-case behaviour but the same worst-case behaviour [DPP04]. Probabilistic minimization has undergone a similarly dynamic development [BEM00, CS02, ZHEJ08, GVdV18], and the best algorithms for minimization of Markov chains now have the same $\mathcal{O}(m \log n)$ run-time as the relational Paige-Tarjan algorithm [HT92, DHS03, VF10]. Using ideas from abstract interpretation, Ranzato and Tapparo [RT08] have developed a relational partition refinement algorithm that is generic over *notions of process equivalence*. As instances, they recover the classical Paige-Tarjan algorithm for strong bisimilarity and an algorithm for stuttering equivalence, and obtain new algorithms for simulation equivalence and for a new process equivalence; the generic run-time analysis, however, is coarser for this algorithm, and in particular does not recover the $\mathcal{O}(m \log n)$ bound for the classical Paige-Tarjan algorithm. Recently, Groote et al. [GJKW17] have presented an improved algorithm for relational partition refinement that covers stuttering, branching and strong bisimilarity.

In this paper we follow an orthogonal approach and provide a generic partition refinement algorithm that can be instantiated for many different *types* of systems (e.g. nondeterministic, probabilistic, weighted). The key to genericity is to use the methods of universal coalgebra. That is, we encapsulate transition types of systems as endofunctors on sets (or a more general category), and model systems as coalgebras for a given type functor.

Overview of the paper. In Section 2, the categorical generalizations of the standard set operations on partitions and equivalence relations are introduced. A short introduction to coalgebras as a framework for state-based systems is given.

In order to explain the generic pattern that existing partition refinement algorithms in the literature follow, we exhibit in Section 3 an informal partition refinement algorithm in natural language that operates on a high level of generality.

In Section 4, the generic pattern is made precise by a categorical construction, in which we work with coalgebras for a monomorphism-preserving endofunctor on a category with image factorizations. Here we present a quite general category-theoretic partition refinement algorithm, and we prove its correctness. The algorithm is parametrized over a select routine

that determines which observations are used to split blocks of states. We present two select routines; one yields a known coalgebraic final-chain algorithm (e.g. [ABH+12, KK14]), the other routine is "select the smaller half", a trick that goes back to Hopcroft [Hop71] and lies at the heart of most modern partition refinement algorithms including Paige and Tarjan's [PT87], being responsible for the logarithmic (rather than linear) dependence of the run-time on the number of states.

While the categorical construction recomputes the involved partitions from scratch in each iteration, we present an optimized version of our algorithm (Section 5) that computes the partitions incrementally. For the correctness of the optimization, we need to restrict to sets and assume that the type endofunctor satisfies a condition we call *zippability*. This property holds, e.g. for all polynomial endofunctors on sets and for the type functors of labelled and weighted transition systems, but is not closed under composition of functors.

In order to be able to provide a concrete presentation of our algorithm and perform a complexity analysis we make the algorithm parametric in an abstract *refinement interface* of the type functor, which encapsulates simple functor specific computations needed to implement the "select the smaller half" routine. In Section 6 we introduce refinement interfaces, and we provide several examples for various zippable type functors of interest and show that they can be implemented with a linear run-time.

Then in Section 7 we provide pseudocode for the algorithm using the incremental computation of the partitions from Section 5 and with the "select the smaller half" routine hard-wired. We show that if the refinement interface operations can be implemented to run in linear time, then the algorithm runs in time $\mathcal{O}((m+n)\cdot \log n)$, where n is the number of states and m the number of 'edges' in a syntactic encoding of the input coalgebra. We thus recover the complexity of the most efficient known algorithms for transition systems (Paige and Tarjan [PT87]), for weighted systems (Valmari and Franceschinis [VF10]), for the task of color refinement (Berkholz, Bonsma, and Grohe [BBG17]), as well as Hopcroft's classical automata minimization algorithm [Hop71] for a fixed alphabet A and $m = n \cdot |A|$.

Section 8 is devoted to modularity and explains how to handle combinations of system types, in particular functor composition. We will see that this can be achieved with just a bit of extra preprocessing, so that our main algorithm need not be adjusted at all. In fact, given a functor T built as a term from finitary functors $\mathsf{Set}^k \to \mathsf{Set}$, we first recall from [SP11] how this induces a functor $\bar{T}\colon \mathsf{Set}^n \to \mathsf{Set}^n$ on multisorted sets, and we present a transformation from finite T-coalgebras to finite $\bar{T}$-coalgebras (with possibly more states) that reflects bisimilarity minimization. We then present a new construction that provides for every functor on the category $\mathcal{C}^n$, where $\mathcal{C}$ is any extensive category (e.g. Set), a functor on $\mathcal{C}$ and a transformation from coalgebras of the former to coalgebras of the latter that preserves the size of carriers (i.e. the number of states) and preserves and reflects bisimilarity minimization. This yields a reduction from bisimilarity minimization of T-coalgebras to minimization of $\coprod\bar{T}\Delta$-coalgebras, where $\Delta\colon \mathsf{Set} \to \mathsf{Set}^n$ is the diagonal functor and $\coprod\colon \mathsf{Set}^n \to \mathsf{Set}$ takes coproducts. The latter problem is solved by the algorithm from Section 7, because we prove that if T is built from functors fulfilling our assumptions, then $\coprod\bar{T}\Delta$ fulfils the assumptions too – even if T itself does not.

As instances of this result, we obtain an efficient *modular* algorithm for systems whose type is built from basic system types fulfilling our assumptions, e.g. probability, nondeterminism, weighted branching (with weights in an arbitrary abelian group), by composition, finite products and finite coproducts (Example 8.19).

One of these instances is an $\mathcal{O}((m+n)\log(m+n))$ algorithm for Segala systems, to our knowledge a new result (more precisely, we improve an earlier bound established by Baier, Engelen, and Majster-Cederbaum [BEM00], roughly speaking by letting only non-zero probabilistic edges enter into the time bound). Note that Groote et al.'s recent algorithm [GVdV18] for Segala systems, which was discovered independently and almost at the same time, has a similar complexity as ours. We also obtain efficient minimization algorithms for general Segala systems and alternating systems [Han94]. In further work [DMSW19] we extend our algorithm to cover weighted branching (with weights in an arbitrary monoids) and tree automata, for which we obtain an algorithm improving the previous best complexity for minimization w.r.t. backwards bisimulation. In addition *op. cit.* presents a generic implementation in the form of the partition refinement tool CoPaR.

This paper is an extended and completely reworked version of a previous conference paper [DMSW17]. Besides providing detailed proofs of all our results, we have included the new Section 8 showing that modularity is achieved without any adjustment of our algorithm.

Acknowledgement. We would like to thank the anonymous referees for their comments, which helped to improve the presentation of our paper.

2. Preliminaries

It is advisable for readers to be familiar with basic category theory [AHS90]. However, our results can be understood by reading the notation in the usual set-theoretic way, which corresponds to their meaning in Set, the category of sets and functions. For the convenience of the reader we recall some concepts that are central for the categorical version of our algorithm.

2.1. Equivalence Relations and Partitions, Categorically.

Our most general setting is a category $\mathcal{C}$ in which we have a well-behaved notion of equivalence relation corresponding to quotient objects. We will assume that $\mathcal{C}$ has finite products and pullbacks.

Notation 2.1. The terminal object of $\mathcal{C}$ is denoted by 1, with unique morphisms $!\colon A \to 1$. In Set, $1 = \{0\}$ as usual. We denote the product of objects A, B by $A \xleftarrow{\pi_1} A \times B \xrightarrow{\pi_2} B$. Given $f\colon D \to A$ and $g\colon D \to B$, the morphism induced by the universal property of the product $A \times B$ is denoted by $\langle f, g\rangle\colon D \to A \times B$.

For morphisms $f\colon A \to D$, $g\colon B \to D$, we denote by

$$\begin{array}{ccc} P & \xrightarrow{\pi_1} & A \\ {\scriptstyle \pi_2}\downarrow & \lrcorner & \downarrow{\scriptstyle f} \\ B & \xrightarrow{g} & D \end{array}$$

that P (together with the projections π_1, π_2) is the pullback of f along g. In Set, we have

$$P = \{(a, b) \in A \times B \mid f(a) = g(b)\}.$$

The *kernel* $\ker f$ of a morphism $f\colon A \to B$ is the pullback of f along itself. We write $\rightarrowtail$ for monomorphisms (in Set, the monomorphisms are the injections), and $\twoheadrightarrow$ for regular epimorphisms; by definition, $q\colon A \to B$ is a regular epimorphism ($q\colon A \twoheadrightarrow B$) if there exists a parallel pair of morphisms $f, g\colon R \rightrightarrows A$ such that q is the coequalizer of f and g. In Set, the coequalizer of $f, g\colon R \rightrightarrows A$ is the quotient of A modulo the smallest equivalence relation on A

relating $f(r)$ and $g(r)$ for all $r \in R$. Here, one can think of R as a set of witnesses r of the pairs $(f(r), g(r))$ generating that equivalence (note that there may be several witnesses for the same pair). Hence, we will often denote the coequalizer of $f, g\colon R \rightrightarrows A$ by $\kappa_R\colon A \twoheadrightarrow A/R$, and such a map represents a *quotient*. When f and g are clear from the context, we will just write $R \rightrightarrows A$.

Kernels and coequalizers allow us to talk about equivalence relations and partitions in a category, and when we speak of an *equivalence relation on the object* X of $\mathcal{C}$ we mean the kernel of some morphism with domain X. Indeed, recall that for every set A, every equivalence relation $\sim$ is the kernel of the canonical quotient map $\kappa_\sim\colon A \twoheadrightarrow A/\!\sim$, and there is a bijection between equivalence relations and partitions on A. In order to obtain a similar bijection for more general categories than Set we make the following global assumption.

Assumption 2.2. We assume throughout that $\mathcal{C}$ is a finitely complete category that has coequalizers and in which regular epimorphisms are closed under composition.

Examples 2.3. Examples of categories satisfying Assumption 2.2 abound. In particular, every *regular* category with coequalizers satisfies our assumptions. The category Set of sets and functions is regular. Every topos is regular, and so is every finitary variety, i.e. a category of algebras for a finitary signature satisfying given equational axioms (e.g. monoids, groups, vector spaces etc.). If $\mathcal{C}$ is regular, so is the functor category $\mathcal{C}^{\mathcal{E}}$ for any category $\mathcal{E}$. For our main applications, we will be interested in the special case $\mathcal{C}^n$ where n is a natural number, i.e. the case where $\mathcal{E}$ is the discrete category with the set of objects $\{1, \dots, n\}$.

The category of posets and the category of topological spaces both fail to be regular but still satisfy our assumptions.

In Set, a function $f\colon A \to B$ factorizes through the partition $A/\mathsf{ker}\, f$ induced by its kernel, via the function $[-]_f\colon A \twoheadrightarrow A/\mathsf{ker}\, f$ taking equivalence classes

$$[x]_f := \{x' \in A \mid f(x) = f(x')\} = \{x' \in D \mid (x, x') \in \mathsf{ker}\, f\}. \tag{2.1}$$

Well-definedness of functions on $A/\mathsf{ker}\, f$ is determined precisely by the universal property of $[-]_f$ as a coequalizer of $\mathsf{ker}\, f \rightrightarrows A$. In particular, f induces an injection $A/\mathsf{ker}\, f \rightarrowtail B$; together with $[-]_f$, this is the factorization of f into a regular epimorphism and a monomorphism.

More generally, our category $\mathcal{C}$ from Assumption 2.2 has a (RegEpi,Mono)-factorization system [AHS90, Prop. 14.22], that is, every morphism $f\colon A \to B$ has a factorization

$$\begin{array}{ccccc} A & & \xrightarrow{\quad f \quad} & & B, \\ & \searrow^{e} & & \nearrow_{m} & \\ & & \mathsf{Im}(f) & & \end{array}$$

where m is a monomorphism and e is a regular epimorphism, specifically the coequalizer of the kernel $\pi_1, \pi_2\colon \mathsf{ker}\, f \rightrightarrows A$. The subobject $m\colon \mathsf{Im}(f) \rightarrowtail B$ is called the *image* and the factorization $f = m \cdot e$ the *image factorization* of f. In every category, we have the *diagonal fill-in property* for monomorphisms m and regular epimorphisms e: Whenever $f \cdot e = m \cdot g$ then there exists a unique diagonal d such that $d \cdot e = g$ and $m \cdot d = f$, implying that (RegEpi,Mono)-factorizations are unique.

Using image factorizations it is easy to show that in our category $\mathcal{C}$, there is a bijection between kernels $K \rightrightarrows A$ and quotients of A – the two directions of this bijection are given by taking the kernel of a coequalizer and by taking the coequalizer of a kernel. In particular, every regular epimorphism is the coequalizer of its kernel.

Furthermore, the above bijection is in fact an order isomorphism between the natural partial orderings on kernels and quotients, respectively. In detail, relations from A to B in $\mathcal{C}$, i.e. jointly monic spans $A \leftarrow R \rightarrow B$, and in particular kernels, represent subobjects of $A \times B$, which are ordered by inclusion in the usual way: We say that a relation $\langle p_1, p_2\rangle \colon R \rightarrowtail A \times B$ (or a kernel) is *finer than* a relation $\langle p_1', p_2'\rangle \colon R' \rightarrowtail A \times B$ if there exists $m \colon R \rightarrowtail R'$ (necessarily unique and monic) such that $p_i' \cdot m = p_i$, for $i = 1, 2$. We write $\cap$ (*intersection*) and $\cup$ (*union*) for meets and joins of kernels in the inclusion ordering on *relations* (not equivalence relations or kernels) on A. In this notation,

$$\ker\langle f, g\rangle = \ker f \cap \ker g; \tag{2.2}$$

in particular, kernels are stable under intersection of relations. Similarly, a quotient represented by $q_1 \colon A \twoheadrightarrow B_1$ is *finer than* a quotient represented by $q_2 \colon A \twoheadrightarrow B_2$ if there exists $b \colon B_1 \twoheadrightarrow B_2$ (necessarily unique and regular epic) with $q_2 = b \cdot q_1$.

We need a few simple observations on kernels that are familiar when instantiated to Set:

Remark 2.4. (1) For every $f \colon X \to Y$ and $g \colon Y \to Z$, $\ker(f)$ is finer than $\ker(g \cdot f)$.

(2) For every $f \colon X \to Y$ and every mono $m \colon Y \rightarrowtail Z$, $\ker(m \cdot f) = \ker f$.

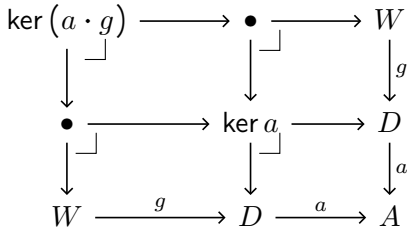
(3) For every $f \colon X \to Y$ and regular epi $q \colon X \twoheadrightarrow Z$, $\ker(f) = \ker(q)$ iff there exists a mono $m \colon Z \rightarrowtail Y$ with $f = m \cdot q$. One implication follows from the previous point, and conversely, take the image factorization $f = n \cdot e$. Then $\ker(e) = \ker(f)$ by point (2), and therefore e and q represent the same quotient, which means there exists some isomorphism $i \colon Z \to \mathsf{Im}(f)$ with $i \cdot q = e$. It follows that $m = n \cdot i$ is the desired mono.

(4) For every $f \colon X \to Z$ and regular epi $e \colon X \twoheadrightarrow Y$, $\ker(e)$ is finer than $\ker(f)$ iff there exists some morphism $g \colon Y \to Z$ such that $g \cdot e = f$.

To see this let $p, q \colon \ker(e) \rightrightarrows X$ be the kernel pair of e. If $\ker(e)$ is finer than $\ker(f)$, then we have $f \cdot p = f \cdot q$. Hence, since e is the coequalizer of its kernel pair we obtain g as desired from the universal property of e. For the other direction apply (1).

(5) Whenever $\ker(a \colon D \to A) = \ker(b \colon D \to B)$ then $\ker(a \cdot g) = \ker(b \cdot g)$, for every $g \colon W \to D$.

Indeed, the kernel $\ker(a \cdot g)$ can be obtained from $\ker a$ by pasting pullback squares as shown below (and similarly for $b \colon D \to B$):



So if $\ker a = \ker b$, then $\ker(a \cdot g) = \ker(b \cdot g)$.

Even though only existence of coequalizers is assumed, $\mathcal{C}$ has more colimits:

Lemma 2.5. *$\mathcal{C}$ has pushouts of regular epimorphisms (i.e. pushouts of spans containing at least one regular epimorphism).*

Proof. Let $X \xleftarrow{e} Y \xrightarrow{h} W$ be a span, with e a regular epi. Let (π_1, π_2) be the kernel pair of e, and let $q \colon W \twoheadrightarrow Z$ be the coequalizer of $h \cdot \pi_1$ and $h \cdot \pi_2$. Since e is the coequalizer of

π_1, π_2 there exists $r\colon X \to Z$ such that $r \cdot e = q \cdot h$. We claim that the square in

$$\begin{array}{ccc} \ker e & \overset{\pi_1}{\underset{\pi_2}{\rightrightarrows}} Y & \xrightarrow{e} \!\!\!\!\rightarrow X \\ & {\scriptstyle h}\downarrow & \downarrow{\scriptstyle r} \\ & W & \xrightarrow[q]{} \!\!\!\!\rightarrow Z \end{array}$$

is a pushout. Uniqueness of mediating morphisms is clear since q is epic; it remains to show existence. So let $W \xrightarrow{g} U \xleftarrow{f} X$ be such that $g \cdot h = f \cdot e$. Then $g \cdot h \cdot \pi_1 = f \cdot e \cdot \pi_1 = f \cdot e \cdot \pi_2 = g \cdot h \cdot \pi_2$, so by the universal property of q we obtain $k\colon Z \to U$ such that $k \cdot q = g$. It remains to check that $k \cdot r = f$. Indeed, we have $k \cdot r \cdot e = k \cdot q \cdot h = g \cdot h = f \cdot e$, which implies the claim because e is epic. □

2.2. **Coalgebra.** We now briefly recall basic notions from universal coalgebra, seen as a unified framework for state-based reactive systems. For introductory texts, see [Rut00, JR97, Adá05, Jac17]. Given an endofunctor $H\colon \mathcal{C} \to \mathcal{C}$, a *coalgebra* is a pair (X, ξ) where X is an object of $\mathcal{C}$ called the *carrier* and thought of as an object of *states*, and $\xi\colon X \to HX$ is a morphism called the *structure* of the coalgebra. Our leading examples are the following.

Example 2.6. (1) Labelled transition systems with labels from a set A are coalgebras for the functor $HX = \mathcal{P}(A \times X)$ (and unlabelled transition systems are simply coalgebras for $\mathcal{P}$) on Set. Explicitly, a coalgebra $\xi\colon X \to HX$ assigns to each state x a set $\xi(x) \in \mathcal{P}(A \times X)$, and this represents the transition structure at x: x has an a-transition to y iff $(a, y) \in \xi(x)$. In concrete examples, we restrict to the finite powerset functor $\mathcal{P}_\mathrm{f} X = \{S \in \mathcal{P}X \mid S \text{ finite}\}$, and coalgebras for $HX = \mathcal{P}_\mathrm{f}(A \times X)$ are finitely branching LTSs.

(2) Weighted transition systems with weights from a commutative monoid $(M, +, 0)$ are modelled as coalgebras as follows. We consider the *monoid-valued* functor $M^{(-)}$ defined on sets X by

$$M^{(X)} = \{f\colon X \to M \mid f(x) \neq 0 \text{ for only finitely many } x\},$$

and on maps $h\colon X \to Y$ by

$$M^{(h)}(f)(y) = \textstyle\sum_{h(x)=y} f(x).$$

M-weighted transition systems are in bijective correspondence with coalgebras for $M^{(-)}$, and for M-weighted labelled transition systems one takes $(M^{(-)})^A$, where A is the label alphabet (see [GS01]).

(3) The finite powerset functor $\mathcal{P}_\mathrm{f}$ is the monoid-valued functor $\mathbb{B}^{(-)}$ for the Boolean monoid $\mathbb{B} = (2, \vee, 0)$. The *bag functor* $\mathcal{B}_\mathrm{f}$, which assigns to a set X the set of bags (i.e. finite multisets) on X, is the monoid-valued functor for the additive monoid of natural numbers $(\mathbb{N}, +, 0)$.

(4) Probabilistic transition systems are modelled coalgebraically using the distribution functor $\mathcal{D}$. This is the subfunctor $\mathcal{D}X \subseteq \mathbb{R}_{\geq 0}^{(X)}$, where $\mathbb{R}_{\geq 0}$ is the monoid of addition on the non-negative reals, given by $\mathcal{D}X = \{f \in \mathbb{R}_{\geq 0}^{(X)} \mid \sum_{x \in X} f(x) = 1\}$.

(5) Simple (resp. general) Segala systems [Seg95] strictly alternate between non-deterministic and probabilistic transitions. Simple Segala systems can be modelled as coalgebras for

the set functor $\mathcal{P}_\mathrm{f}(A \times \mathcal{D}(-))$, which means that for every label $a \in A$, a state non-deterministically proceeds to one of a finite number of possible probability distributions over states. General Segala systems are coalgebras for $\mathcal{P}_\mathrm{f}\mathcal{D}(A \times -)$, which means that a state s non-deterministically proceeds to one of a finite number of distributions over the set of A-labelled transitions from s.

(6) Let Σ be a signature (a.k.a ranked alphabet), i.e. a set of (operation) symbols, each with a prescibed natural number, its *arity*. The corresponding *polynomial functor* $H_\Sigma\colon \mathsf{Set} \to \mathsf{Set}$ maps a set X to the set

$$H_\Sigma X = \coprod_{n\in\mathbb{N}} \Sigma_n \times X^n,$$

where Σ_n is the set of symbols of arity n, and H_Σ acts similarly on maps. Note that the elements of $H_\Sigma X$ may be identified with shallow terms over X, i.e. formal expressions $\sigma(x_1, \dots, x_n)$, where $\sigma \in \Sigma$ is an n-ary symbol and $x_1, \dots, x_n \in X$.

A coalgebra $\xi\colon X \to H_\Sigma X$ is a deterministic system, where the coalgebra structure assigns to each state $x \in X$ a tuple $\xi(x) = (\sigma, x_1, \dots, x_n)$ in which $\sigma \in \Sigma$ is an n-ary *output* symbol and $x_1, \dots, x_n$ are successor states, one for every input in $\{1, \dots, n\}$. Coalgebras for H_Σ can also be thought of as top-down deterministic tree automata.

(7) For a fixed finite set A, the coalgebras of the functor $HX = 2 \times X^A$ are deterministic automata for the input alphabet A (neglecting the initial state). Concretely, a coalgebra structure $\langle o, \delta\rangle\colon X \to 2 \times X^A$ consists of the characteristic function $o\colon X \to 2$ of the set of final states of the automaton and the next state function δ. Note that the functor H is (naturally isomorphic to) a polynomial functor, where the signature Σ consists of two A-ary operation symbols.

A *coalgebra morphism* from a coalgebra (X, ξ) to a coalgebra (Y, ζ) is a morphism $h\colon X \to Y$ such that $\zeta \cdot h = Hh \cdot \xi$; intuitively, coalgebra morphisms preserve observable behaviour. Coalgebras and their morphisms form a category $\mathsf{Coalg}(H)$. The forgetful functor $\mathsf{Coalg}(H) \to \mathcal{C}$ creates all colimits, so $\mathsf{Coalg}(H)$ has all colimits that $\mathcal{C}$ has; in particular, our running assumptions imply the following:

$$\begin{array}{ccc} X & \xrightarrow{\xi} & HX \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle Hh} \\ Y & \xrightarrow{\zeta} & HY \end{array}$$

Corollary 2.7. $\mathsf{Coalg}(H)$ *has all coequalizers and pushouts of regular epimorphisms.*

A *subcoalgebra* of a coalgebra (X, ξ) is represented by a coalgebra morphism $m\colon (Y, \zeta) \rightarrowtail (X, \xi)$ such that m is a monomorphism in $\mathcal{C}$. Likewise, a *quotient* of a coalgebra (X, ξ) is represented by a coalgebra morphism $q\colon (X, \xi) \twoheadrightarrow (Y, \zeta)$ carried by a regular epimorphism q of $\mathcal{C}$. If H preserves monomorphisms, then the image factorization structure on $\mathcal{C}$ lifts to coalgebras in the sense that every coalgebra morphism f has a factorization $f = m \cdot e$ into coalgebra morphisms m and e such that m is a monomorphism and e a regular epimorphism in $\mathcal{C}$ (see e.g. [MPW19, Lemma 2.5]).

Recall that a coalgebra is called *simple* if it does not have any non-trivial quotients [Ihr03]. We will use the following equivalent characterization:

Proposition 2.8. *If H preserves monomorphisms, then a coalgebra (X, ξ) is simple iff every coalgebra morphism with domain (X, ξ) is carried by a monomorphism.*

Proof. For necessity, consider a coalgebra morphism $h\colon (X, \xi) \to (Y, \zeta)$ and take its image factorization to obtain $q\colon (X, \xi) \twoheadrightarrow (\mathsf{Im}(h), i)$ and $m\colon (\mathsf{Im}(h), i) \rightarrowtail (Y, \zeta)$ in $\mathsf{Coalg}(H)$. Since (X, ξ) is simple, q is an isomorphism, and so $h = m \cdot q$ is a monomorphism. For sufficiency,

consider $q\colon (X,\xi) \twoheadrightarrow (Y,\zeta)$ with q a regular epimorphism in $\mathcal{C}$. By assumption, q is also monic, whence an isomorphism. □

Intuitively, in a simple coalgebra all states exhibiting the same observable behaviour are already identified. This paper is concerned with the design of algorithms for computing *the* simple quotient of a given coalgebra:

Lemma 2.9. *A simple quotient of a coalgebra is unique (up to isomorphism). Concretely, let (X,ξ) be a coalgebra, and let $e_i\colon (X,\xi) \twoheadrightarrow (Y_i,\zeta_i)$, $i = 1,2$, be quotients with (D_i,d_i) simple. Then (D_1,d_1) and (D_2,d_2) are isomorphic; more precisely, e_1 and e_2 represent the same quotient.*

Proof. By Corollary 2.7, there is a pushout $Y_1 \xrightarrow{f_1} Z \xleftarrow{f_2} Y_2$ of $Y_1 \xleftarrow{e_1} X \xrightarrow{e_2} Y_2$ in $\mathsf{Coalg}(H)$. Since regular epimorphisms are generally stable under pushouts, f_1 and f_2 are regular epimorphisms, hence isomorphisms because Y_1 and Y_2 are simple; this proves the claim. □

Existence of the simple quotient can be shown under additional assumptions on $\mathcal{C}$ (cf. Remark 2.12 below):

Theorem 2.10 [Ihr03]**.** *Assume that $\mathcal{C}$ is cocomplete and cowellpowered. Then every coalgebra (X,ξ) has a simple quotient given by the cointersection (i.e. the wide pushout) of all quotient coalgebras*

$$q\colon (X,\xi) \twoheadrightarrow (X',\xi').$$

For $\mathcal{C} = \mathsf{Set}$, two elements $x \in X$ and $y \in Y$ of coalgebras (X,ξ) and (Y,ζ) are *behaviourally equivalent* if they can be merged by coalgebra morphisms, that is, if there exist a coalgebra (Z,ω) and coalgebra morphisms $f\colon (X,\xi) \to (Z,\omega)$, $g\colon (Y,\zeta) \to (Z,\omega)$ such that $f(x) = g(y)$. Intuitively, the simple quotient of a coalgebra in Set is its quotient modulo behavioural equivalence. In our main examples, this means that we minimize w.r.t. standard bisimilarity-type equivalences:

Example 2.11. Behavioural equivalence instantiates to various notions of bisimilarity:

(1) Park-Milner bisimilarity on labelled transition systems [AM89];

(2) weighted bisimilarity on weighted transition systems [Kli09, Proposition 2];

(3) stochastic bisimilarity on probabilistic transition systems [Kli09];

(4) Segala bisimilarity on simple and general Segala systems [BSdV03, Theorem 4.2].

Remark 2.12. A *final coalgebra* is a terminal object in the category of coalgebras, i.e. a coalgebra (T,τ) such that every coalgebra (X,ξ) has a unique coalgebra morphism into (T,τ). There are reasonable conditions under which a final coalgebra is guaranteed to exist, e.g. when $\mathcal{C}$ is a locally presentable category (in particular, when $\mathcal{C} = \mathsf{Set}$) and H is accessible [AR94]. If H preserves monomorphisms and has a final coalgebra (T,τ), then the simple quotient of a coalgebra (X,ξ) is the image of (X,ξ) under the unique morphism into (T,τ); in particular, in this case every coalgebra has a simple quotient.

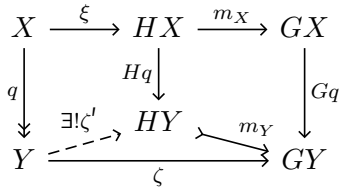
Finally, we note a useful result that implies that computing the simple quotient of a coalgebra for H can be reduced to computing the simple quotient of its induced coalgebra for a superfunctor of H:

Proposition 2.13. *Suppose that $m\colon H \rightarrowtail G$ is a natural transformation with monomorphic components. Then every H-coalgebra $\xi\colon X \to HX$ and its induced G-coalgebra*

$$X \xrightarrow{\xi} HX \xrightarrow{m_X} GX$$

have the same quotients and, hence, the same simple ones (if they exists).

Proof. We prove only the first claim. Let $q\colon X \twoheadrightarrow Y$ be a regular epimorphism. It suffices to show that q carries an H-coalgebra morphism with domain (X, ξ) iff it carries a G-coalgebra morphism with domain $(X, m_X \cdot \xi)$. Note that $m\colon H \rightarrowtail G$ induces an embedding $\mathsf{Coalg}(H) \to \mathsf{Coalg}(G)$. Hence, 'only if' is clear, and we prove 'if'. Suppose that q is a coalgebra morphism $(X, m_X \cdot \xi) \to (Y, \zeta)$. Then the outside of the following diagram commutes:



By the naturality of m the right-hand part commutes. Hence, since m_Y is monic we obtain ζ' as in the diagram by the diagonal fill-in property (Section 2.1), making q an H-coalgebra morphism $(X, \xi) \twoheadrightarrow (Y, \zeta')$. □

3. Partition Refinement from an Abstract Point of View

In the next section we will provide an abstract partition refinement algorithm and formally prove its correctness. Our main contribution is *genericity*: we are able to state and prove our results at a level of abstraction that uniformly captures various kinds of state based systems. This allows us to instantiate our efficient generic algorithm to many different (combinations of) transition structures. Before describing the abstract algorithm (Algorithm 4.9) formally, we now give an informal description of a partition refinement algorithm, which will make it clear which parts of partition refinement algorithms are generic and which parts are specific to a particular transition type. Although Algorithm 4.9 works in categorical generality, we use set-theoretic parlance in the present informal discussion.

Given a system with a set X of states, one of the core ideas of the known partition refinement algorithms mentioned so far, in particular the algorithms by Hopcroft [Hop71] and Paige-Tarjan [PT87], is to maintain two equivalence relations P and Q on X, represented by the corresponding partitions X/P and X/Q, where X/P will be "one transition step ahead of X/Q", so the relation P is a refinement of Q. Therefore, the elements of X/P are called *subblocks* and the elements of X/Q are called *compound blocks*.

Initially, we put $X/Q = \{X\}$ and let X/P be the initial partition with respect to the "output behaviour" of the states in X. For example, in the case of deterministic automata, the output behaviour of a state is its finality, i.e. the initial partition separates final from non-final states; in the case of transition systems, the initial partition separates deadlock states from states with successors; and for weighted systems, the initial partition groups the states by the sum of weights of outgoing edges.

Algorithm 3.1 (Informal Partition Refinement)**.** Given a system on X and initial partitions $X/Q = \{X\}$ and X/P as above, iterate the following while P is different from Q:

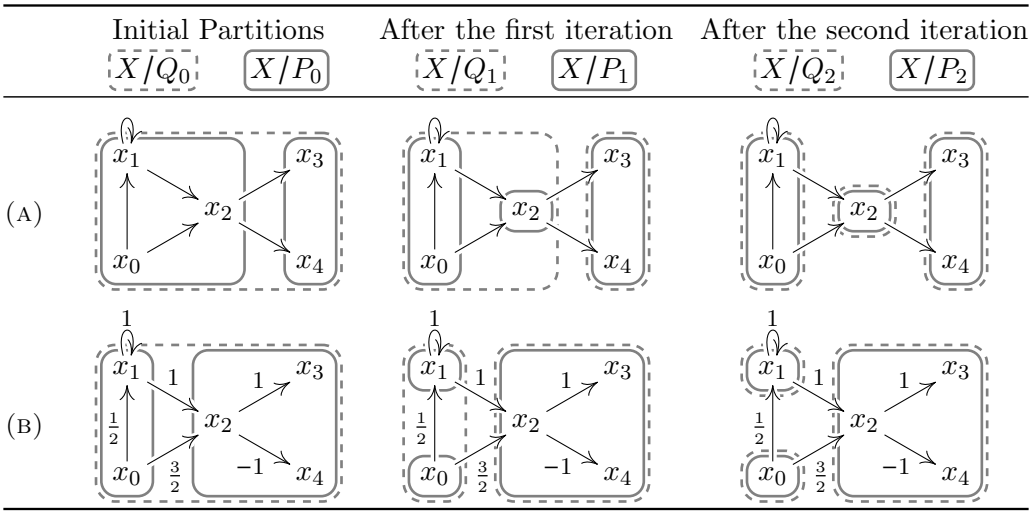


FIGURE 1. Example of the partition refinement in (A) transition systems ($H = \mathcal{P}_\mathrm{f}$) and (B) $\mathbb{R}$-weighted systems ($H = \mathbb{R}^{(-)}$). The partition X/Q_i is indicated by dashed lines, and X/P_i by solid lines.

(1) Pick a subblock S in X/P that is properly contained in a compound block $C \in X/Q$, i.e. $S \subsetneq C$. Note that this choice represents a quotient $q\colon X \twoheadrightarrow \{S, C \setminus S, X \setminus C\}$.

(2) Refine X/Q with respect to this quotient by splitting C into two blocks S and $C \setminus S$; that is, replace X/Q with the intersection (greatest common refinement) of the partitions X/Q and $\{S, C \setminus S, X \setminus C\}$ according to the terminology introduced in Section 2.

(3) Update X/P to the coarsest refinement of X/P in which any two states are distinguished if the transition structure distinguishes them up to Q, equivalently: If two states are identified, then the transition structure identifies them up to Q.

Remark 3.2. (1) Since X/Q is refined using only information from X/P in each iteration, the algorithm maintains the invariant that the partition X/P is finer than X/Q.

(2) The informal property of the refinement of X/P in step (3) is referred to as *stability* of P w.r.t. Q. The formal definition of this notion takes into account the transition type of the given system. In fact, Paige and Tarjan [PT87] have defined stability for transition systems, and we generalize their definition to the level of coalgebras in Definition 4.3.

Note that Steps (1) and (2) are independent of the given transition type, as they perform only basic operations on quotients. In contrast, the initialization procedure and Step (3) depend on the transition type of the specific system, encoded by the type functor.

Example 3.3. Figure 1 illustrates runs of partition refinement algorithms and how the transition type specific steps are handled for two different types of systems: in Paige and Tarjan's algorithm for transition systems [PT87] and in Valmari and Franceschinis' Markov chain lumping algorithm, which in fact works for $\mathbb{R}$-weighted systems [VF10]:

(A) In transition systems, the initial partition only distinguishes between deadlocks and live states; in our example $X/P_0 = \{\{x_0, x_1, x_2\}, \{x_3, x_4\}\}$, and $X/Q_0 = \{X\}$ identifies all states.

- In the first iteration of the loop of Algorithm 3.1, there are two choices for S in Step (1), $\{x_0, x_1, x_2\}$ and $\{x_3, x_4\}$, both leading to the same refinement step: In Step (2) the only

block of X/Q_0 is split into two blocks, and so $X/Q_1 = X/P_0$. In Step (3), X/P_1 is the coarsest refinement of X/P_0 that is stable w.r.t. X/Q_1. In transition systems, X/P is stable w.r.t. X/Q if for all elements in the same block $x, x' \in B \in X/P$ and any other block $B' \in X/Q$, x has an edge to B' iff x' has an edge to B' [PT87]. In detail, the state x_2 becomes a separate block because x_2 has a transition to $\{x_3, x_4\} \in X/Q_1$ whereas x_0 and x_1 do not. The states x_0 and x_1 remain in the same block in X/P_1, because both have a transition to $\{x_0, x_1, x_2\} \in X/Q_1$, and have no transition to $\{x_3, x_4\} \in X/Q_1$. So $X/P_1 = \{\{x_0, x_1\}, \{x_2\}, \{x_3, x_4\}\}$.

- In the second iteration, we have symmetric choices $\{x_0, x_1\}$ and $\{x_2\}$ for S, and both split the compound block $\{x_0, x_1, x_2\}$ so that $X/Q_2 = X/P_1$. The partition X/Q_2 is stable w.r.t. itself because both x_0 and x_1 have transitions to $\{x_0, x_1\}$ and $\{x_2\}$, and because x_3 and x_4 both are deadlock states (for the singleton block $\{x_2\}$ there is nothing to check). Hence, $X/P_2 = X/Q_2$ and the algorithm terminates.

(B) In weighted systems, the initial partition groups the states by the sum of the weights of their outgoing transitions.[1] For x_0 and x_1, the sum is 2, and for x_2, x_3, x_4 the sum is 0. Hence $X/P_0 = \{\{x_0, x_1\}, \{x_2, x_3, x_4\}\}$, and as always, $X/Q_0 = \{X\}$ identifies all states.

- In the first iteration of Algorithm 3.1, we can choose $\{x_0, x_1\}$ or $\{x_2, x_3, x_4\}$ for S, both leading to $X/Q_1 = X/P_0$. In weighted systems, stability of X/P w.r.t. X/Q means that all elements of the same block $x, x' \in B \in X/P$ have the same accumulated transition weight to any other block $B' \in X/Q$, for $f\colon X \to \mathbb{R}^{(X)}$ this means $\sum_{y \in B'} f(x)(y) = \sum_{y \in B'} f(x')(y)$ (called 'compatibility' in [VF10]). When refining X/P_1 to be as coarse as possible and stable w.r.t. X/Q_1, we have the following accumulated sums. The transition from x_2 to $\{x_2, x_3, x_4\} \in X/Q_1$ has weight 0, just like the (non-existent) transition from x_3 (resp. x_4) to $\{x_2, x_3, x_4\} \in X/Q_1$, and hence x_2, x_3, x_4 are identified in X/P_1. The transition from x_0 to $\{x_2, x_3, x_4\} \in X/Q_1$ has weight $\frac{3}{2}$, but the transition from x_1 to $\{x_2, x_3, x_4\}$ has weight 1. Thus, x_0 and x_1 are split in X/P_1, and therefore we have $X/P_1 = \{\{x_0\}, \{x_1\}, \{x_2, x_3, x_4\}\}$.
- In the second iteration, we can choose S to be $\{x_0\}$ or $\{x_1\}$. In either case, $X/Q_2 = X/P_1$, and we have that X/Q_2 is stable w.r.t. X/Q_2 because the transition from x_2 to $\{x_2, x_3, x_4\}$ in X/Q_2 has weight 0. Thus, $X/P_2 = X/Q_2$ and the algorithm terminates.

In the remainder of the paper we shall see that one can unify the similarities between Figure 1(A) and Figure 1(B) into a generic algorithm on a coalgebraic level, where the specifics of the transition type are hidden within the coalgebraic type functor.

4. A Categorical Algorithm for Behavioural Equivalence

We proceed to give a formal description of a categorical partition refinement algorithm that computes the simple quotient of a given coalgebra under fairly general assumptions.

[1] For actual Markov chains or probabilistic transition systems, where weights sum up to 1, the initial partition would thus be trivial, causing immediate termination; this corresponds to the fact that all states of a Markov chain are behaviourally equivalent unless we introduce additional observable features such as propositional atoms or deadlock. In standard treatments of Markov chain lumping, such additional features are abstracted in the choice of a non-trivial initial partition.

Assumption 4.1. In addition to Assumption 2.2, we fix an endofunctor $H\colon \mathcal{C} \to \mathcal{C}$ that preserves monomorphisms.

Remark 4.2. For $\mathcal{C} = \mathsf{Set}$, the assumption that H preserves monomorphisms is w.l.o.g. First note that every endofunctor on Set preserves nonempty monomorphisms. Moreover, for every set functor H there exists a mono-preserving set functor H' that is identical to H on the full subcategory of all nonempty sets [AT90, Theorem 3.4.5]. Hence, H' has essentially the same coalgebras as H since there is only one coalgebra structure on $\varnothing$.

For a given coalgebra $\xi\colon X \to HX$ in Set, any partition refinement algorithm should maintain a quotient $q\colon X \twoheadrightarrow X/Q$ that distinguishes some (but possibly not all) states with different behaviour, and in fact, initially q typically identifies everything. Using the language of universal coalgebras, one can express the transition type specific steps from Algorithm 3.1 generically. Informally, our algorithm repeats the following steps until it stabilizes:

(1) Analyse $X \xrightarrow{\xi} HX \xrightarrow{Hq} H(X/Q)$ to identify equivalence classes w.r.t. q (i.e. compound blocks) containing states that exhibit distinguishable behaviour when considering one more step of the transition structure ξ.

(2) Use parts of this information to refine q.

Here, the composite $Hq \cdot \xi$ (more precisely its kernel) defines the finer partition X/P in Algorithm 3.1 (our algorithm will guarantee that this does refine the previous value of P). More precisely, P will be defined to be the kernel of $Hq \cdot \xi$, and so P is as coarse as possible to be stable w.r.t. $q\colon X \twoheadrightarrow X/Q$, in the following sense:

Definition 4.3. Given a coalgebra $\xi\colon X \to HX$, a kernel $P \rightrightarrows X$ is said to be *stable* w.r.t. a morphism $q\colon X \to Y$ provided that there exists a morphism $\xi/q\colon X/P \to HY$ such that the following square commutes:

$$\begin{array}{ccc} X & \xrightarrow{\xi} & HX \\ {\scriptstyle \kappa_P}\downarrow & & \downarrow{\scriptstyle Hq} \\ X/P & \xrightarrow{\xi/q} & HY \end{array}$$

Equivalently, the kernel $\pi_1, \pi_2\colon P \rightrightarrows X$ is stable w.r.t. q if $Hq \cdot \xi \cdot \pi_1 = Hq \cdot \xi \cdot \pi_2$.

Remark 4.4. Note that $P \rightrightarrows X$ is stable w.r.t. $\kappa_P\colon X \twoheadrightarrow X/P$ iff κ_P is a coalgebra morphism and thus represents a quotient of (X, ξ).

Example 4.5. This definition of stability for a coalgebra $\xi\colon X \to HX$ matches the concrete instances we have seen in Example 3.3:

(1) In a transition system $\xi\colon X \to \mathcal{P}_{\mathrm{f}}X$, $P \rightrightarrows X$ is stable w.r.t. $\kappa_Q\colon X \twoheadrightarrow X/Q$ if for all elements in the same block $x, x' \in B \in X/P$ and any other block $B' \in X/Q$, x has an edge to B' iff x' has an edge to B' [PT87].

(2) In a weighted system $\xi\colon X \to \mathbb{R}^{(X)}$, stability of $P \rightrightarrows X$ w.r.t. X/Q means that all elements of the same block $x, x' \in B \in X/P$ have the same accumulated transition weight to any other block $B' \in X/Q$, i.e. $\sum_{y \in B'} \xi(x)(y) = \sum_{y \in B'} \xi(x')(y)$. This is called 'compatibility' in [VF10].

(3) For LTSs, Blom and Orzan [BO05] define the *signature* of state $x \in X$ with respect to q as $Hq \cdot \xi(x)$ (with $HX = \mathcal{P}(A \times X)$) and then define a partition X/P to be stable if every two members $(x, x') \in P$ have the same signature w.r.t. $q := \kappa_P$.

The refinement step (2) above corresponds to the subblock selection in Algorithm 3.1. This selection will be encapsulated at the present level of generality in a routine select, assumed as a parameter of our algorithm:

Definition 4.6. A select routine is an operation that receives a chain of two regular epis $X \xrightarrow{y} Y \xrightarrow{z} Z$ and returns some morphism $\mathsf{select}(y,z)\colon Y \to K$. We refer to Y and Z as the objects of *subblocks* and *compound blocks*, respectively.

In our algorithm, y will represent the canonical quotient $X \twoheadrightarrow X/P$ and z the canonical map $X/P \twoheadrightarrow X/Q$ given by the invariant that P is finer than Q. Intuitively, the morphism $\mathsf{select}(y,z)\colon Y \to K$ selects some of the information contained in Y and discards all the remaining information by identifying the remaining elements. For example, in the Paige-Tarjan algorithm with $Y = X/P$ and $Z = X/Q$, select models the selection of only a single block $S \in X/P$ leading to a split of the surrounding block $C \in X/Q, S \subseteq C$ into two blocks S and $C \setminus S$. In this case, $\mathsf{select}(y,z)$ is thus essentially a characteristic function of the following shape:

Definition 4.7. For sets $S \subseteq C \subseteq X$, define the map

$$\chi_S^C\colon X \to 3 \qquad 3 = \{0,1,2\} \qquad \chi_S^C(x) = \begin{cases} 2 & \text{if } x \in S \\ 1 & \text{if } x \in C \setminus S \\ 0 & \text{if } x \in X \setminus C. \end{cases}$$

This is a three-valued version of the characteristic function $\chi_M\colon X \to 2$ for a subset $M \subseteq X$. Put differently, χ_S^C is the codomain restriction of $\langle \chi_S, \chi_C \rangle\colon X \to 2 \times 2$ obtained by leaving out the impossible case of $x \in S \setminus C$.

Example 4.8. We present some examples of select routines. Throughout, we fix a chain $X \xrightarrow{y} Y \xrightarrow{z} Z$ of quotients.

(1) In Hopcroft's algorithm [Hop71], and in all the known efficient partition refinement algorithms mentioned so far, the goal is to find a proper subblock that is at most half the size of the compound block it is contained in. The optimized version of our algorithm over $\mathcal{C} = \mathsf{Set}$ (Section 7) will use the same strategy, embodied in the following select routine. First, we identify the elements of Y with the corresponding equivalence classes in $X/\ker y$ and those of Z with equivalence classes in $X/\ker(z \cdot y)$. Now let S be a subblock, i.e. $S \in X/\ker y \cong Y$, such that its compound block, i.e. $C = z(S) \in X/\ker(z \cdot y) \cong Z$ satisfies $2 \cdot |S| \leq |C|$. Then we put (using our notation (2.1) for equivalence classes induced by the map z):

$$\mathsf{select}(y,z) = \chi_{\{S\}}^{[S]_z}\colon Y \to 3. \tag{4.1}$$

Note that we then have

$$\mathsf{select}(y,z) \cdot y = \chi_S^C\colon X \to 3.$$

If there is no such $S \in Y$, then X is infinite or z is an isomorphism, and in either case we simply put $\mathsf{select}(y,z) = \mathsf{id}_Y$.

(2) One obvious choice for $\mathsf{select}(y,z)\colon Y \to K$ is the identity on Y, so that *all* of the information present in Y is used for further refinement. We will explain in Remark 4.16 how under this choice, our algorithm instantiates to König and Küpper's final chain algorithm [KK14].

(3) Two other, trivial, choices are $\mathsf{select}(y,z)\colon Y \xrightarrow{!} 1$ and $\mathsf{select}(y,z) = z$. Since both of these choices discard all the information in Y, this will leave the partitions computed by the algorithm unchanged, see the proof of Theorem 4.20.

Given a select routine, the most general form of our partition refinement algorithm works as follows. Given a coalgebra $\xi\colon X \to HX$, we successively refine equivalence relations (i.e. kernel pairs) $Q \rightrightarrows X$ and $P \rightrightarrows X$, maintaining the invariant that P is finer than Q (cf. Lemma 4.11), which is witnessed by a (necessarily unique) morphism $f\colon X/P \twoheadrightarrow X/Q$ such that $f \cdot \kappa_P = \kappa_Q$, where κ_P and κ_Q denote the canonical quotients (i.e. coequalizers) of the above two kernel pairs (Section 2.1).

Before each iteration of the main loop, we take into account new information on the behaviour of states, represented by a morphism $q\colon X \to K$, and accumulate this information in a morphism $\bar{q}\colon X \to \bar{K}$ where $\bar{K}$ is the Cartesian product of the instances of K encountered up to the present iteration. In order to facilitate the analysis later, we index the variables $P, Q, f, q, \bar{q}$ over loop iterations i in the description. For brevity, we will just write the objects Q_i and P_i in lieu of the respective kernel pairs $P_i \rightrightarrows X$ and $Q_i \rightrightarrows X$.

Algorithm 4.9. Given a coalgebra $\xi\colon X \to HX$ and a select routine, initially put

$$Q_0 = X \times X, \qquad \bar{q}_0 = !\colon X \to 1 = K_0, \qquad P_0 = \ker(X \xrightarrow{\xi} HX \xrightarrow{H!} H1).$$

Then iterate the following steps, with i counting iterations starting at 0, while P_i is properly finer than Q_i:

(1) $q_{i+1} := X \xrightarrow{\kappa_{P_i}} X/P_i \xrightarrow{\mathsf{select}(\kappa_{P_i}, f_i)} K_{i+1}, \quad \bar{q}_{i+1} := \langle \bar{q}_i, q_{i+1} \rangle\colon X \longrightarrow \prod_{j \le i} K_i \times K_{i+1}$

where $f_i\colon X/P_i \twoheadrightarrow X/Q_i$ witnesses that P_i is finer than Q_i.

(2) $Q_{i+1} := \ker \bar{q}_{i+1} = Q_i \cap \ker q_{i+1}$

(3) $P_{i+1} := \ker\big(X \xrightarrow{\xi} HX \xrightarrow{H\bar{q}_{i+1}} H \prod_{j \le i+1} K_j \big)$

Upon termination of the above loop return $X/P_i = X/Q_i$.

Remark 4.10. Note that Algorithm 4.9 is precisely the informal Algorithm 3.1 where

- the partitions and equivalence relations are replaced by coequalizers and kernel pairs;
- P_i being properly finer than Q_i means that the canonical morphism $f_i\colon X/P_i \twoheadrightarrow X/Q_i$ is not an isomorphism;
- in Step (2), we have $\ker \bar{q}_{i+1} = \ker\langle \bar{q}_i, q_{i+1} \rangle = \ker \bar{q}_i \cap \ker q_{i+1} = Q_i \cap \ker q_{i+1}$, see (2.2);
- transition type specific steps involve how the type functor H acts on morphisms;
- Step (3) makes P_{i+1} stable w.r.t. $\bar{q}_{i+1}$;
- the choice of a subblock S and a compound block C is performed by the select routine.

In general, Algorithm 4.9 need not terminate, but we present sufficient conditions for termination in Theorem 4.20. We now proceed to show that when Algorithm 4.9 terminates, then it returns the (carrier object of the) simple quotient of (X, ξ), i.e. we prove correctness. We continue to use the notation established in Algorithm 4.9. Since $\bar{q}$ accumulates more information in every step, it is clear that P and Q are being successively refined:

Lemma 4.11. *For every i, we have monomorphisms $P_{i+1} \rightarrowtail P_i \rightarrowtail Q_{i+1} \rightarrowtail Q_i$ witnessing inclusions of relations.*

(The above inclusions state that the kernel $P_{i+1} \rightrightarrows X$ is finer than the kernel $P_i \rightrightarrows X$ etc., see Section 2.1.)

Proof. We have that Q_{i+1} is finer than Q_i by definition and use Remark 2.4(1) for the remaining inclusions:

(1) $P_{i+1} \rightarrowtail P_i$: Let $p\colon \prod_{j\le i+1} K_j \to \prod_{j\le i} K_j$ be the product projection. We have $H\bar{q}_i \cdot \xi = Hp \cdot H\bar{q}_{i+1} \cdot \xi$, so $P_{i+1} = \mathsf{ker}(H\bar{q}_{i+1} \cdot \xi)$ is finer than $P_i = \mathsf{ker}(H\bar{q}_i \cdot \xi)$.

(2) $P_i \rightarrowtail Q_{i+1}$: First observe that for every i, P_i is finer than $\mathsf{ker}\, q_{i+1}$ because q_{i+1} factors through $\kappa_{P_i}\colon X \twoheadrightarrow X/P_i$ by step (1) in Algorithm 4.9. We now obtain the desired result by induction on i. In the base case we have that P_0 is finer than $Q_1 = \mathsf{ker}\langle \bar{q}_0, q_1\rangle = \mathsf{ker}\langle !, q_1\rangle = \mathsf{ker}\, q_1$. For the induction step ($i > 0$), since $Q_{i+1} = Q_i \cap \mathsf{ker}\, q_{i+1}$, it suffices to show that P_i is finer than Q_i and $\mathsf{ker}\, q_{i+1}$. The latter is just our lead-in observation, and for the former use the inductive hypothesis ($P_{i-1} \rightarrowtail Q_i$) and the fact that, by (1), P_i is finer than P_{i-1}. □

One of the key ingredients in the correctness proof is that the partition X/P_i is one "transition step" ahead of X/Q_i, i.e. the kernel P_i is stable w.r.t. the quotient κ_{Q_i}:

Proposition 4.12. *There exist monomorphisms $\xi/Q_i\colon X/P_i \rightarrowtail H(X/Q_i)$ for $i \ge 0$ (necessarily unique) such that* (4.2) *commutes.*

$$\begin{array}{ccc} X & \xrightarrow{\xi} & HX \\ {\scriptstyle \kappa_{P_i}}\downarrow & & \downarrow{\scriptstyle H\kappa_{Q_i}} \\ X/P_i & \xrightarrow{\xi/Q_i} & H(X/Q_i) \end{array} \tag{4.2}$$

Proof. Since $Q_i = \mathsf{ker}\, \bar{q}_i$, the image factorization of $\bar{q}_i$ has the form

$$\bar{q}_i = \big(X \xrightarrow{\kappa_{Q_i}} X/Q_i \xrightarrow{m} \textstyle\prod_{j\le i} K_j \big).$$

By definition of P_i and because H preserves monos, we thus have $P_i = \mathsf{ker}(H\bar{q}_i \cdot \xi) = \mathsf{ker}(H\kappa_{Q_i} \cdot \xi)$, and hence we obtain ξ/Q_i as in (4.2) by the universal property of the coequalizer κ_{P_i}:

$$\begin{array}{ccccc} P_i & & & & \\ \downarrow\downarrow & & & & \\ X & \xrightarrow{\xi} & HX & & \\ {\scriptstyle \kappa_{P_i}}\downarrow & & \downarrow{\scriptstyle H\kappa_{Q_i}} & \searrow^{H\bar{q}_i} & \\ X/P_i & \xrightarrow{\xi/Q_i} & H(X/Q_i) & \xrightarrow{Hm} & H\big(\textstyle\prod_{j\le i} K_j\big) \end{array}$$

In fact, κ_{P_i} is the regular-epi part of the factorization of $H\bar{q}_i \cdot \xi$, and so $Hm \cdot \xi/Q_i$ is the mono part, and thus ξ/Q_i is also a monomorphism. □

Corollary 4.13. *If $P_i = Q_i$ for some i, then X/Q_i carries a unique coalgebra structure making κ_{P_i} a coalgebra morphism.*

For $\mathcal{C} = \mathsf{Set}$, this means that all states of X that are merged by the algorithm are actually behaviourally equivalent. We still need to prove the converse, namely that all behaviourally equivalent states are indeed identified in X/Q_i:

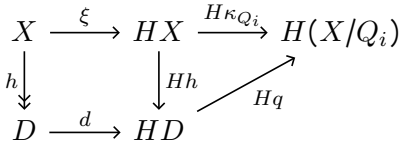
Theorem 4.14 (Correctness)**.** *If $P_i = Q_i$ for some i, then $\xi/Q_i\colon X/Q_i \to H(X/Q_i)$ is a simple coalgebra.*

Proof. Let $h\colon (X,\xi) \twoheadrightarrow (D,d)$ represent a quotient.

(1) We first prove that for all $i \geq 0$,

$$\text{if } \ker h \text{ is finer than } Q_i\text{, then } \ker h \text{ is finer than } P_i. \tag{4.3}$$

This is seen as follows: If $\ker h$ is finer than Q_i, then $\kappa_{Q_i}\colon X \twoheadrightarrow X/Q_i$ factorizes through $h\colon X \twoheadrightarrow D$, i.e. we have some $q\colon D \twoheadrightarrow X/Q_i$ such that $q \cdot h = \kappa_{Q_i}$ (see Remark 2.4(4)). So $H\kappa_{Q_i} \cdot \xi$ factorizes through $Hh \cdot \xi$ and hence through h, since $Hh \cdot \xi = d \cdot h$:



Since $P_i = \ker(H\kappa_{Q_i} \cdot \xi)$, this implies that $\ker h$ is finer than P_i, again by Remark 2.4(4).

(2) Next we prove by induction on i that $\ker h$ is finer than both P_i and Q_i, for all $i \geq 0$. For $i = 0$, the claim for $Q_0 = X \times X$ is trivial, and the one for P_0 follows by (4.3). In the induction step, we have by the inductive hypothesis that $\ker(h)$ is finer than P_i, thus by Lemma 4.11 also finer than Q_{i+1} and consequently by (4.3) finer than P_{i+1}.

(3) Now we are ready to prove the claim of the theorem. Let $q\colon (X/Q_i, \xi/Q_i) \twoheadrightarrow (D,d)$ represent a quotient. Then $q \cdot \kappa_{Q_i}\colon (X,\xi) \to (D,d)$ represents a quotient of (X,ξ), so by point (2) above, $\ker(q \cdot \kappa_{Q_i})$ is finer than Q_i. By Remark 2.4(1), $Q_i = \ker(\kappa_{Q_i})$ is also finer than $\ker(q \cdot \kappa_{Q_i})$, so $\ker(q \cdot \kappa_{Q_i}) = \ker(\kappa_{Q_i}) = Q_i$. This implies that $\kappa_{Q_i}\colon X \twoheadrightarrow X/Q_i$ is the regular epi part of the image factorization of $q \cdot \kappa_{Q_i}$, i.e. we have $m \cdot \kappa_{Q_i} = q \cdot \kappa_{Q_i}$ for some monomorphism m. Since κ_{Q_i} is an epimorphism, we obtain $m = q$, i.e. q is a monomorphism, and hence an isomorphism. □

Remark 4.15. Most classical partition refinement algorithms are parametrized by an initial partition $\kappa_{\mathcal{I}}\colon X \twoheadrightarrow X/\mathcal{I}$. We start with the trivial partition $!\colon X \to 1$ because a non-trivial initial partition might split equivalent behaviours and then would invalidate Theorem 4.14. To accommodate an initial partition $X/\mathcal{I}$ coalgebraically, replace (X,ξ) with the coalgebra $\langle \xi, \kappa_{\mathcal{I}} \rangle$ for the functor $H(-) \times X/\mathcal{I}$ – indeed, already P_0 will then be finer than $\mathcal{I}$.

We look in more detail at two corner cases of the algorithm where the select routine retains all available information, respectively none.

Remark 4.16. If $\mathsf{select}(X \xrightarrow{y} Y \xrightarrow{z} Z\ ) = \mathsf{id}_Y$ (cf. Example 4.8(2)), then Algorithm 4.9 becomes König and Küppers' final chain algorithm [KK14], as we will now explain.

(1) Recall that H induces the *final chain*:

$$1 \xleftarrow{!} H1 \xleftarrow{H!} H^2 1 \xleftarrow{H^2!} \cdots \xleftarrow{H^{i-1}!} H^i 1 \xleftarrow{H^i!} H^{i+1}1 \xleftarrow{H^{i+1}!} \cdots$$

(The chain is transfinite but we consider only the first ω stages.) Every coalgebra $\xi\colon X \to HX$ then induces a *canonical cone* $\xi^{(i)}\colon X \to H^i 1$ on the final chain, defined inductively by

$$\xi^{(0)} = !\colon X \to H^0 1 = 1 \quad \text{and} \quad \xi^{(i+1)} = (X \xrightarrow{\xi} HX \xrightarrow{H\xi^{(i)}} HH^i 1 = H^{i+1}1).$$

The objects $H^n 1$ may be thought of as domains of n-step behaviour for H-coalgebras. If $\mathcal{C} = \mathsf{Set}$ and X is finite, then states x and y are behaviourally equivalent iff $\xi^{(i)}(x) = \xi^{(i)}(y)$ for all $i < \omega$ [Wor05]. In fact, Worrell showed this for unrestricted X and for *finitary* set functors H, i.e. set functors preserving filtered colimits; equivalently, H is finitary if for every

$x \in HX$ there exists a finite subset $m\colon Y \hookrightarrow X$ and $y \in HY$ such that $x = Hm(y)$. Note that for a *finite* coalgebra for an arbitrary set functor H, behavioural equivalence remains the same when we pass to the *finitary part* of H, i.e. the functor given by

$$H_f X = \bigcup \{Hm[Y] \mid m\colon Y \hookrightarrow X \text{ and } Y \text{ finite}\}.$$

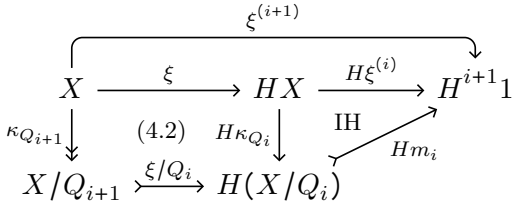
To see this note that if two states in a finite coalgebra can be identified by a coalgebra morphism into some H-coalgebra, then they can be identified by a coalgebra morphism into a finite H-coalgebra. This is just by image factorization of coalgebras (see Section 2.2).

(2) The inclusions $P_i \rightarrowtail Q_{i+1}$ in Lemma 4.11 reflect that only some and not necessarily all of the information present in the relation P_i (resp. the quotient X/P_i) is used for further refinement. If indeed everything is used, then $Q_{i+1} = P_i$, and our algorithm simply computes the kernels of the morphisms $\xi^{(i)}\colon X \to H^i 1$ forming the canonical cone:

Proposition 4.17. *If* $\mathsf{select}(X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z) = \mathsf{id}_Y$, *then for all* $i \in \mathbb{N}$, $Q_i = \ker \xi^{(i)}$.

Proof. With $\mathsf{select}(y, z) = \mathsf{id}_Y$, we have $q_{i+1} = \kappa_{P_i}\colon X \to X/P_i$, and so $\ker q_{i+1} = P_i$ for all $i \in \mathbb{N}$. Thus, $\ker q_{i+1}$ is finer than Q_i by Lemma 4.11. It follows that $Q_{i+1} = Q_i \cap \ker q_{i+1} = \ker q_{i+1} = P_i$.

In order to prove that $Q_i = \ker \xi^{(i)}$, for all $i \in \mathbb{N}$, we construct monomorphisms $m_i\colon X/Q_i \rightarrowtail H^i 1$ with $m_i \cdot \kappa_{Q_i} = \xi^{(i)}$ inductively (which implies $Q_i = \ker \xi^{(i)}$ by Remark 2.4.(2)). For $i = 0$, we trivially have $m_0\colon X/Q_0 \xrightarrow{\cong} 1$. In the inductive step, we put $m_{i+1} := Hm_i \cdot \xi/Q_i$:



□

Intuitively, the select routine in Proposition 4.17 retains all available information. The other extreme is the following:

Definition 4.18. We say that select is *discarding* at $X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z$ if $\mathsf{select}(y, z)\colon Y \to K$ factorizes through z. Further, we call select *progressing* if $\mathsf{select}(y, z)$ is discarding at y, z only if z is an isomorphism.

Example 4.19. (1) The select picking the smaller half in Example 4.8(1) is progressing. We prove the contraposition: if z in $X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z$ is not an isomorphism, let $S \in X/\ker y \cong Y$ be the subblock used in the definition (4.1) of $\mathsf{select}(y, z)$, and note that then there also exists a subblock $B \in X/\ker y \cong Y$ with $z(B) = z(S)$ and $|S| \leq |B|$. By the definition of $k = \mathsf{select}(y, z)$, we have $k(B) = 1 \neq 2 = k(S)$, and so k cannot factor through z.

(2) The select routine that always returns id_Y for $X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z$ in Example 4.8(2) is trivially progressing: if id_Y factorizes through z, then z is a (split) mono, and hence an isomorphism.

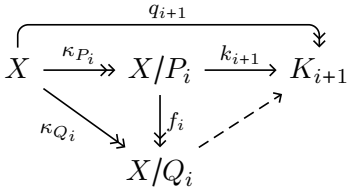
(3) The select routine that returns the morphism $!\colon Y \to 1$ or $z\colon Y \to Z$ is always discarding, and thus fails to be progressing (unless all regular epis in $\mathcal{C}$ are isomorphisms).

Theorem 4.20. *If* select *is progressing, then Algorithm 4.9 terminates and computes the simple quotient of the input coalgebra* (X, ξ)*, provided that* (X, ξ) *has only finitely many quotients.*

E.g. for $\mathcal{C} = \mathsf{Set}$, every finite coalgebra has only finitely many quotients.

Proof. (1) We first show that our algorithm fails to progress in the $(i+1)^{\text{st}}$ iteration, i.e. $Q_{i+1} = Q_i$, iff select is discarding at $X/P_i, X/Q_i$, i.e. iff $k_i := \mathsf{select}\big(X \overset{\kappa_{P_i}}{\twoheadrightarrow} X/P_i \overset{f_i}{\twoheadrightarrow} X/Q_i\big)$ factorizes through f_i.

To see this, first note that select is discarding at $X/P_i, X/Q_i$ iff q_{i+1} factorizes through κ_{Q_i}:



We thus have the desired equivalence: q_{i+1} factorizes through κ_{Q_i} iff (by Remark 2.4(4)) Q_i is finer than $\ker q_{i+1}$ iff $Q_i = Q_i \cap \ker q_{i+1} = Q_{i+1}$.

(2) We proceed to prove the claim. Lemma 4.11 shows that we obtain a chain of successively finer quotients X/Q_i. Since X has only finitely many quotients, there must be an i such that $Q_i = Q_{i+1}$, and this implies, using point (1), that select is discarding at $X/P_i, X/Q_i$. Since the select routine is progressing, we obtain $P_i = Q_i$ as desired. □

5. Incremental Partition Refinement

In the most generic version of the partition refinement algorithm (Algorithm 4.9), the partitions are recomputed from scratch in every step: In Step (3) of the algorithm, $P_{i+1} = \ker(H\langle \bar{q}_i, q_{i+1}\rangle \cdot \xi)$ is computed from the information $\bar{q}_i$ accumulated so far and the new information q_{i+1}, but in general one cannot exploit that the kernel of $\bar{q}_i$ has already been computed. We now present a refinement of the algorithm in which the partitions are computed incrementally, i.e. P_{i+1} is computed from P_i and q_{i+1}. This requires the type functor H to be *zippable* (Definition 5.1) and the select routine to *respect compound blocks* (Definition 5.14).

Note that in Step (2), Algorithm 4.9 computes a kernel $Q_{i+1} = \ker \bar{q}_{i+1} = \ker\langle \bar{q}_i, q_{i+1}\rangle$ as the intersection of $\ker(\bar{q}_i)$ and $\ker(q_i)$ (cf. (2.2)). Hence, the partition $X/\ker \bar{q}_{i+1}$ for such a kernel can be computed in two steps:

(1) Compute $X/\ker \bar{q}_i$.

(2) Refine every block in $X/\ker \bar{q}_i$ with respect to $q_{i+1} \colon X \to K_{i+1}$.

Algorithm 4.9 can thus be implemented to keep track of the partition X/Q_i and then refine this partition by q_{i+1} in each iteration.

However, the same trick cannot be applied immediately to the computation of X/P_i, because of the functor H inside the computation of the kernel: $P_{i+1} = \ker(H\langle \bar{q}_i, q_{i+1}\rangle \cdot \xi)$. In Proposition 5.18, we will provide sufficient conditions for H, $a \colon D \to A$, $b \colon D \to B$ to satisfy

$$\ker H\langle a, b\rangle = \ker\langle Ha, Hb\rangle.$$

As soon as this holds for $a = \bar{q}_i, b = q_{i+1}$, we can optimize the algorithm by changing Step (3) to

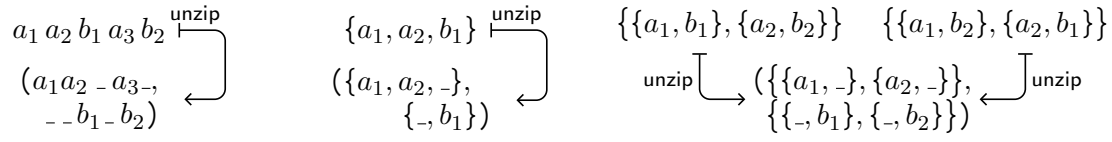
$$P'_{i+1} := \mathsf{ker}\langle H\bar{q}_i \cdot \xi, Hq_{i+1} \cdot \xi\rangle \quad (= P_i \cap \mathsf{ker}(Hq_{i+1} \cdot \xi)). \tag{5.1}$$

The conditions on a and b will be ensured by a condition on select, and the condition on the functor H is as follows:

Definition 5.1. A functor $H\colon C \to \mathcal{D}$ is *zippable* if the following morphisms are monomorphisms for every objects A and B:

$$\mathsf{unzip}_{H,A,B}\colon H(A+B) \xrightarrow{\langle H(A+!),H(!+B)\rangle} H(A+1) \times H(1+B)$$

Intuitively, if H is a functor on Set, we may think of elements t of $H(A+B)$ as shallow terms with variables from $A+B$. Then zippability means that each t is uniquely determined by the two terms obtained by replacing A- and B-variables, respectively, by some placeholder $\_$, viz. the element of 1, as illustrated in the examples in Figure 2.



(A) $(-)^*$ is zippable (B) $\mathcal{P}_\mathrm{f}$ is zippable (C) $\mathcal{P}_\mathrm{f}\mathcal{P}_\mathrm{f}$ is not zippable

FIGURE 2. Zippability of Set-Functors for sets $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2\}$.

Lemma 5.2. *Let H be zippable and $f\colon A \to C$, $g\colon B \to D$. Then the following is a mono:*

$$H(A+B) \xrightarrow{\langle H(A+g),H(f+B)\rangle} H(A+D) \times H(C+B)$$

Proof. By finality of 1, the diagram

$$\begin{array}{ccc} H(A+B) & & \\ {\scriptstyle \langle H(A+g),H(f+B)\rangle}\downarrow & \searrow^{\mathsf{unzip}_{H,A,B} = \langle H(A+!),H(!+B)\rangle} & \\ H(A+D) \times H(C+B) & \xrightarrow{H(A+!)\times H(!+B)} & H(A+1) \times H(1+B) \end{array}$$

commutes. Since the diagonal arrow is monic, so is $\langle H(A+g), H(f+B)\rangle$. □

Assumption 5.3. For the remainder of Section 5, we assume that $\mathcal{C} = \mathsf{Set}$.

However, most proofs are category-theoretic to clarify where working in Set is really needed and where the arguments are more general.

Example 5.4. (1) Constant functors $X \mapsto A$ are zippable: unzip is the diagonal $A \to A \times A$.
(2) The identity functor is zippable since $\langle A+!, !+B\rangle\colon A+B \to (A+1)\times(1+B)$ is monic in Set.
(3) From Lemma 5.5 it follows that every polynomial endofunctor is zippable. Indeed, a polynomial functor is precisely one that is build from constant and the identity functors using (finite) products and coproducts (cf. Example 2.6(6)).

Lemma 5.5. *Zippable endofunctors are closed under (possibly infinite) products, coproducts and subfunctors.*

(Recall that products and coproducts of functors are formed pointwise, e.g. $(F + G)(X) = FX + GX$.)

Proof. For the closure under products and coproducts, we only provide the proof for the binary case; the proof for arbitrary products and coproducts is completely analogous. Let F, G be endofunctors.

(1) Suppose that both F and G are zippable. That $F \times G$ is zippable follows from monos being closed under products:

$$\begin{array}{ccc} F(A+B) \times G(A+B) & \xrightarrow{\mathsf{unzip}_{F,A,B} \times \mathsf{unzip}_{G,A,B}} & F(A+1) \times F(1+B) \times G(A+1) \times G(1+B) \\ & \searrow_{\mathsf{unzip}_{F\times G,A,B}} & \wr\downarrow \\ & & \big(F(A+1) \times G(A+1)\big) \times \big(F(1+B) \times G(1+B)\big). \end{array}$$

(2) Suppose again that F and G are zippable. To see that $F + G$ is zippable consider the diagram

$$\begin{array}{ccc} F(A+B) + G(A+B) & \xrightarrow{\mathsf{unzip}_{F,A,B} + \mathsf{unzip}_{G,A,B}} & \big(F(A+1) \times F(1+B)\big) + \big(G(A+1) \times G(1+B)\big) \\ & \searrow_{\mathsf{unzip}_{F\times G,A,B}} & \downarrow{\scriptstyle \langle(\pi_1+\pi_1),(\pi_2+\pi_2)\rangle} \\ & & \big(F(A+1) + G(A+1)\big) \times \big(F(1+B) + G(1+B)\big). \end{array}$$

The horizontal morphism is monic since monos are closed under coproducts in Set. The vertical morphism is monic since for any sets A_i and B_i, $i = 1, 2$, the following morphism clearly is a monomorphism:

$$(A_1 \times B_1) + (A_2 \times B_2) \xrightarrow{\langle(\pi_1+\pi_1),(\pi_2+\pi_2)\rangle} (A_1 + A_2) \times (B_1 + B_2).$$

(3) Suppose now that F is a subfunctor of G via $s\colon F \rightarrowtail G$, where G is zippable. Then the following diagram shows that F is zippable, too:

$$\begin{array}{ccc} F(A+B) & \xrightarrow{\mathsf{unzip}_{F,A,B}} & F(A+1) \times F(1+B) \\ \downarrow{\scriptstyle s_{A\times B}} & & \downarrow{\scriptstyle s_{A+1} \times s_{1+B}} \\ G(A+B) & \xrightarrow{\mathsf{unzip}_{G,A,B}} & G(A+1) \times G(1+B). \end{array}$$

Indeed, since the composition of the lower and left-hand morphisms is monomorphic, so is the upper morphism. □

Lemma 5.6. *If H has a componentwise monic natural transformation $H(X + Y) \rightarrowtail HX \times HY$, then H is zippable.*

Proof. Let $\alpha_{X,Y}\colon H(X + Y) \rightarrowtail HX \times HY$ be monic and natural in X and Y. Then the square

$$\begin{array}{ccc} H(A+B) & \xrightarrow{\mathsf{unzip}=\langle H(A+!),H(!+B)\rangle} & H(A+1) \times H(1+B) \\ \downarrow{\scriptstyle \alpha_{A,B}} & & \downarrow{\scriptstyle \alpha_{A,1} \times \alpha_{1,B}} \\ HA \times HB & \xrightarrow{\langle HA\times H!,H!\times HB\rangle} & (HA \times H1) \times (H1 \times HB) \end{array}$$

commutes by naturality of α. The bottom morphism is monic because it has a left inverse, $\pi_1 \times \pi_2$. Therefore, unzip is monic as well. □

Example 5.7. (1) For every commutative monoid, the monoid-valued functor $M^{(-)}$ (see Example 2.6(2)) admits a natural isomorphism $M^{(X+Y)} \cong M^{(X)} \times M^{(Y)}$, and hence is zippable by Lemma 5.6.

(2) As special cases of monoid-valued functors we obtain that the finite powerset functor $\mathcal{P}_\mathrm{f}$ and the bag functor $\mathcal{B}_\mathrm{f}$ are zippable.

(3) By Lemma 5.6, the full powerset functor $\mathcal{P}$ is zippable.

(4) The distribution functor $\mathcal{D}$ (see Example 2.6) is a subfunctor of the monoid-valued functor $\mathbb{R}_{\geq 0}^{(-)}$ for the additive monoid $\mathbb{R}_{\geq 0}$ of real numbers, and hence is zippable by Item (1) and Lemma 5.5.

(5) The previous examples together with the closure properties in Lemma 5.5 show that a number of functors of interest are zippable, e.g. $2 \times (-)^A$, $2 \times \mathcal{P}_\mathrm{f}(-)^A$, $\mathcal{P}_\mathrm{f}(A \times (-))$, $2 \times \big((-) + 1\big)^A$, and variants where $\mathcal{P}_\mathrm{f}$ is replaced by $\mathcal{B}_\mathrm{f}$, $M^{(-)}$, or $\mathcal{D}$.

Remark 5.8. Out of the above results, only zippability of the identity and coproducts of zippable functors make use of properties of Set (Assumption 5.3). Indeed, zippable functors on a category $\mathcal{C}$ are closed under coproducts as soon as monomorphisms are closed under coproducts in $\mathcal{C}$, which is satisfied in most categories of interest. Zippability of the identity holds whenever $\mathcal{C}$ is extensive, i.e. it has well-behaved set-like coproducts (see e.g. [CLW93] or 8.12 later). Examples of extensive categories are the categories of sets, posets and graphs as well as any presheaf category. We will take a closer look at extensive categories when we discuss multisorted coalgebras (Section 8).

Example 5.9. The monotone neighbourhood functor, which maps a set X to the set

$$\mathcal{M}(X) = \{N \subseteq \mathcal{P}X \mid A \in N \wedge B \supseteq A \implies B \in N\},$$

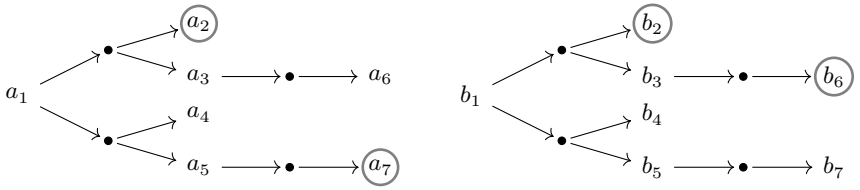
of monotone neighbourhood systems over X, is not zippable, that is, there are distinct monotone neighbourhood systems that are identified by unzip. Indeed, denoting the upwards closure of a set system by $(-)\uparrow$, we have

$$\begin{aligned}\mathsf{unzip}\,(\{\{a_1, b_1\}, \{a_2, b_2\}\}\uparrow) &= \big(\{\{a_1, \_\}, \{a_2, \_\}\}\uparrow, \{\{\_, b_1\}, \{\_, b_2\}\}\uparrow\big)\\ &= \mathsf{unzip}\,(\{\{a_1, b_2\}, \{a_2, b_1\}\}\uparrow)\,.\end{aligned}$$

Example 5.10. The functor $\mathcal{P}_\mathrm{f}\mathcal{P}_\mathrm{f}$ fails to be zippable, as shown in Figure 2. First, this shows that zippable functors are not closed under quotients, since $\mathcal{P}_\mathrm{f}\mathcal{P}_\mathrm{f}$ is a quotient of the polynomial, hence zippable, functor HH where $HX = \coprod_{n<\omega} X^n$. Secondly, this shows that zippable functors are not closed under composition.

The following example shows that the optimized algorithm, i.e. Algorithm 4.9 run with (5.1) in lieu of Step (3), is not correct for the non-zippable functor $\mathcal{P}_\mathrm{f}\mathcal{P}_\mathrm{f}$, even though the select routine used here (see Example 4.8(1)) behaves sufficiently well (specified later in Definition 5.14 and cf. Corollary 5.21).

Example 5.11. Consider the following coalgebra $\xi\colon X \to HX$ for $HX = 2 \times \mathcal{P}_\mathrm{f}\mathcal{P}_\mathrm{f}X$:



Final states, i.e. states x with $\pi_1(\xi(x)) = 1$, are indicated by a circle. Let us replace step (3) of Algorithm 4.9 by equation (5.1), i.e. we compute

$$P_{i+1} = P'_{i+1} \overset{(5.1)}{=} \ker\langle H\bar{q}_i \cdot \xi, Hq_{i+1} \cdot \xi\rangle = \ker P'_i \cap \ker(Hq_{i+1} \cdot \xi).$$

We will show that the states a_1 and b_1 are identified by all P'_i and Q_i, i.e. they are not distinguished by the algorithm, although they are clearly behaviourally inequivalent.

We simplify the partitions by defining abbreviations for the final and non-final states without successors as well as the rest,

$$F := \{a_2, a_7, b_2, b_6\}, \quad N := \{a_4, a_6, b_4, b_7\} \quad \text{and} \quad R := \{a_1, a_3, a_5, b_1, b_3, b_5\}.$$

Then we run the optimized algorithm with the select routine in Example 4.8(1), computing Q_i and P'_i (see (5.1)), and we obtain the following sequence of partitions.

i	q_i	X/Q_i	X/P'_i
0	$!\colon X \to 1$	$\{X\}$	$\{F, N, R\}$
1	$\kappa_{P'_0}\colon X \twoheadrightarrow X/P'_0$	$\{F, N, R\}$	$\{F, N, \{a_1, b_1\}, \{a_3, b_5\}, \{a_5, b_3\}\}$
2	$\chi^{R}_{\{a_1,b_1\}}\colon X \to 3$	$\{F, N, \{a_1, b_1\}, \{a_3, b_5, a_5, b_3\}\}$	$\{F, N, \{a_1, b_1\}, \{a_3, b_5\}, \{a_5, b_3\}\}$
3	$\chi^{\{a_3,b_5,a_5,b_3\}}_{\{a_3,b_5\}}\colon X \to 3$	$\{F, N, \{a_1, b_1\}, \{a_3, b_5\}, \{a_5, b_3\}\}$	$\{F, N, \{a_1, b_1\}, \{a_3, b_5\}, \{a_5, b_3\}\}$

For the subblock $S = \{a_3, b_5\}$ selected in the third iteration we see that $\{a_1, b_1\}$ is not split in X/P'_3 because:

$$\begin{aligned} H\chi^{\{a_3,b_5,a_5,b_3\}}_{\{a_3,b_5\}} \cdot \xi(a_1) &= H\chi^{\{a_3,b_5,a_5,b_3\}}_{\{a_3,b_5\}}\{\{a_2, a_3\}, \{a_4, a_5\}\} \\ &= \{\{0, 2\}, \{0, 1\}\} \\ &= \{\{0, 1\}, \{0, 2\}\} \\ &= H\chi^{\{a_3,b_5,a_5,b_3\}}_{\{a_3,b_5\}}\{\{b_2, b_3\}, \{b_4, b_5\}\} = H\chi^{\{a_3,b_5,a_5,b_3\}}_{\{a_3,b_5\}} \cdot \xi(b_1) \end{aligned}$$

At this point the algorithm terminates because $X/Q_2 = X/P_2$, while incorrectly not distinguishing a_1 and b_1.

Note that this result remains the same if we chose the subblock $\{a_5, b_3\}$ in the third iteration or if we chose $\{a_3, b_5\}$ in the second iteration and $\{a_1, b_1\}$ in the third one.

Observe that, in general, $\ker H\langle a, b\rangle$ differs from $\ker\langle Ha, Hb\rangle$ even if H is zippable:

Example 5.12. For $H = \mathcal{P}_\mathrm{f}$ and product projections $\pi_1\colon A \times B \to A$ and $\pi_2\colon A \times B \to B$, $\langle \mathcal{P}_\mathrm{f}\pi_1, \mathcal{P}_\mathrm{f}\pi_2\rangle$ in general fails to be injective although $\mathcal{P}_\mathrm{f}\langle \pi_1, \pi_2\rangle = \mathcal{P}_\mathrm{f}\mathsf{id}_{A\times B} = \mathsf{id}_{\mathcal{P}_\mathrm{f}(A\times B)}$. Thus

$$\ker \mathcal{P}_\mathrm{f}\langle \pi_1, \pi_2\rangle \cong \mathcal{P}_\mathrm{f}(A \times B) \not\cong \ker\langle \mathcal{P}_\mathrm{f}\pi_1, \mathcal{P}_\mathrm{f}\pi_2\rangle.$$

(A) Q_0, P_0 for $\bar{q}_0 = !$ (B) Q_1, P_1 for $\bar{q}_1 = \kappa_{P_0}$

FIGURE 3. Partitions of a coalgebra ξ for $H = \{\blacktriangle, \blacksquare, \bullet\} \times \mathcal{P}_\mathrm{f}(-)$. X/Q_i is indicated by dashed, X/P_i by solid lines.

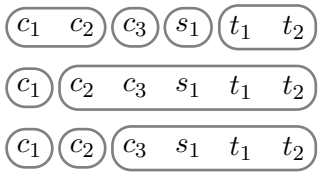


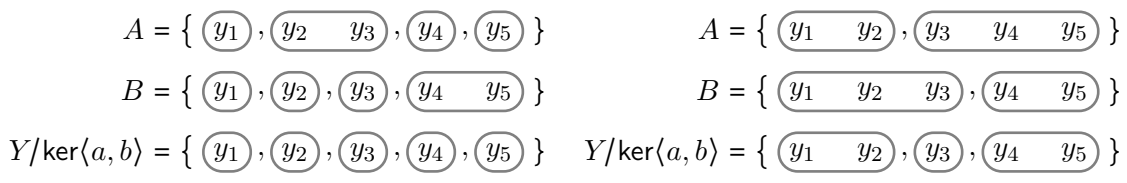
FIGURE 4. Grouping of elements when $S := \{c_1\}$ is chosen as the next subblock and $C := \{c_1, c_2\}$ as the compound block.

Hence, in addition to zippability of H, we will need to enforce constraints on the select routine to achieve the desired optimization (5.1).

The next example illustrates this issue, and a related one: One might be tempted to implement splitting by a subblock S by using the usual characteristic function $q_i = \chi_S \colon X \to K_i$. While this approach is sufficient for systems with real-valued weights [VF10], it may in general let $\ker(H\langle \bar{q}_i, q_{i+1}\rangle \cdot \xi)$ and $\ker\langle H\bar{q}_i \cdot \xi, Hq_{i+1} \cdot \xi\rangle$ differ even if H is zippable, thus rendering the algorithm incorrect:

Example 5.13. Consider the coalgebra $\xi \colon X \to HX$ for the zippable functor $H = \{\blacktriangle, \blacksquare, \bullet\} \times \mathcal{P}_\mathrm{f}(-)$ illustrated in Figure 3 (essentially a Kripke model). The initial partition X/P_0 splits the set of all states by shape and by $\mathcal{P}_\mathrm{f}!$, i.e. states with successors are distinguished from the ones without successors (Figure 3a). Now, suppose that select returns $k_1 := \mathsf{id}_{X/P_0}$, i.e. retains all information (cf. Remark 4.16), so that $Q_1 = P_0$ and P_1 puts c_1 and c_2 into different blocks (Figure 3b). Since $q_0 = !$, we have $\ker \bar{q}_1 = \ker q_1$ and thus simplify notation by directly defining $\bar{q}_1 := \kappa_{P_0}$. We now analyse the next partition that arises when we split w.r.t. the subblock $S = \{c_1\}$ but not w.r.t. the rest $C \setminus S$ of the compound block $C = \{c_1, c_2\}$; in other words, we take $k_2 := \chi_{\{\{c_1\}\}} \colon X/P_1 \to 2$, making $q_2 = \chi_{\{c_1\}} \colon X \to 2$. Then, $H\langle \bar{q}_1, q_2\rangle \cdot \xi$ splits t_1 from t_2, because t_1 has a successor c_2 with $\bar{q}_1(c_2) = \{c_1, c_2\}$ and $q_2(c_2) = 0$ whereas t_2 has no such successor. However, t_1, t_2 fail to be split by $\langle H\bar{q}_1, Hq_2\rangle \cdot \xi$ because their successors do not differ when we look at successor blocks in X/Q_1 and $X/\ker \chi_S$ separately: both have $\{c_1, c_2\}$ and $\{c_3\}$ as successor blocks in X/Q_1 and $\{c_1\}$ and $X \setminus \{c_1\}$ as successors in $X/\ker \chi_S$ (cf. Figure 4). Formally:

$$H\bar{q}_1 \cdot \xi(t_1) = (\mathsf{id} \times \mathcal{P}_\mathrm{f}\kappa_{P_0}) \cdot \xi(t_1) = (\blacktriangle, \{\{c_1, c_2\}, \{c_3\}\}) = H\bar{q}_1 \cdot \xi(t_2),$$
$$Hq_2 \cdot \xi(t_1) = (\mathsf{id} \times \mathcal{P}_\mathrm{f}\chi_{\{c_1\}}) \cdot \xi(t_1) = (\blacktriangle, \{0, 1\}) = Hq_2 \cdot \xi(t_2).$$



(A) $\ker a \cup \ker b$ is transitive. (B) $\ker a \cup \ker b$ is not transitive.

FIGURE 5. Maps $a\colon Y \to A$ and $b\colon Y \to B$, where the elements of A and B are considered as equivalence classes of elements of Y, defining a and b implicitly. $Y \,/\, \ker\langle a, b\rangle$ is the block-wise intersection of the partitions defined by a and b.

So if we computed P_2 iteratively as in (5.1) for $q_2 = \chi_S$, then t_1 and t_2 would not be split, and we would reach the termination condition $P_2 = Q_2$ before all behaviourally inequivalent states have been separated.

Already Paige and Tarjan [PT87, Step 6 of the Algorithm] note that one additionally needs to split by $C \setminus S = \{c_3\}$, which is accomplished by splitting by $q_i = \chi_S^C$ (see Example 4.8(1)). This is formally captured by the condition we introduce next and explain further in Lemma 5.15.

Definition 5.14. A select routine *respects compound blocks* if whenever $k = \mathsf{select}(X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z)$ then the union $\ker z \cup \ker k$ is a kernel.

Since in Set, reflexive and symmetric relations are closed under unions, the definition boils down to $\ker z \cup \ker k$ being transitive. In Set, we have an intuitive characterization:

Lemma 5.15. *For maps $a\colon Y \to A$, $b\colon Y \to B$, the following are equivalent:*

(1) $\ker a \cup \ker b \rightrightarrows Y$ *is a kernel (i.e. an equivalence relation).*

(2) $\ker a \cup \ker b \rightrightarrows Y$ *is the kernel of the pushout of a and b.*

(3) *For all $x, y, z \in Y$, $a(x) = a(y)$ and $b(y) = b(z)$ implies $a(x) = a(y) = a(z)$ or $b(x) = b(y) = b(z)$.*

(4) *For all $x \in Y$, $[x]_a \subseteq [x]_b$ or $[x]_b \subseteq [x]_a$.*

The last item states that when we move from a-equivalence classes to b-equivalence classes, the classes either merge or split, but do not merge with other classes and split at the same time. In Figure 5a, $\ker a \cup \ker b$ is transitive, and thus also a kernel. For $x \in \{y_1, y_2, y_3\}$, we have $[x]_b \subseteq [x]_a$ and for $x \in \{y_4, y_5\}$ we have $[x]_a \subseteq [x]_b$. On the other hand in Figure 5b, $\ker a \cup \ker b$ is not transitive because $(y_1, y_3) \in \ker b$ and $(y_3, y_5) \in \ker a$, but $(y_1, y_5) \notin \ker a \cup \ker b$, and indeed condition (4) of Lemma 5.15 fails because $[y_3]_a \not\subseteq [y_3]_b$ and $[y_3]_b \not\subseteq [y_3]_a$ (because $[y_3]_a = \{y_3, y_4, y_5\}$, $[y_3]_b = \{y_1, y_2, y_3\}$).

In Example 5.13 of a concrete run of our algorithm, one sees in Figure 4 that $Q_1 \cup \ker \chi_S$ fails to be transitive, while $Q_1 \cup \ker \chi_S^C$ is transitive.

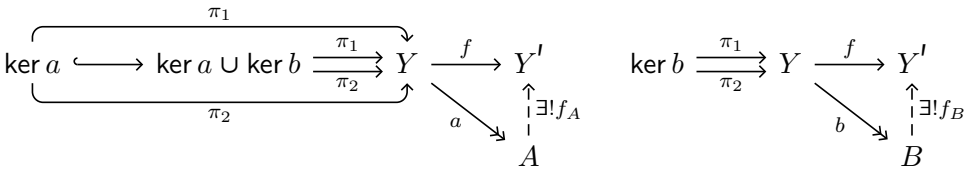
Proof of Lemma 5.15. (4) $\Rightarrow$ (1) In Set, kernels are equivalence relations. Obviously, $\ker a \cup \ker b$ is both reflexive and symmetric. For transitivity, take $(x, y), (y, z) \in \ker a \cup \ker b$. Then $x, z \in [y]_a \cup [y]_b$. If $[y]_a \subseteq [y]_b$, then $x, z \in [y]_b$ and $(x, z) \in \ker b$; otherwise $(x, z) \in \ker a$.

(1) $\Rightarrow$ (2) In Set, monomorphisms are stable under pushouts, so it is sufficient to show that $\ker a \cup \ker b$ is the kernel of the pushout of the regular epis from the image factorization of a and b, respectively. In other words, w.l.o.g. we may assume that a and b are surjective

maps, and we need to check that $\ker a \cup \ker b$ is the kernel of $p := p_A \cdot a = p_B \cdot b$, where p_A and p_B are the two injections of the pushout below:

$$\begin{array}{ccc} Y & \xrightarrow{a} & A \\ {\scriptstyle b}\downarrow & & \downarrow{\scriptstyle p_A} \\ B & \xrightarrow{p_B} & P. \end{array}$$

Let $\ker a \cup \ker b$ be the kernel of some $f\colon Y \to Y'$. Then, f makes the projections of $\ker a$ (resp. $\ker b$) equal and hence the coequalizer a (resp. b) induces a unique f_A (resp. f_B) such that the triangles in the diagrams below commute:



Since $f_B \cdot b = f = f_A \cdot a$, (f_A, f_B) is a competing cocone for the above pushout. This induces a cocone morphism $m\colon (P, p_A, p_B) \to (Y, f_A, f_B)$, and we have

$$m \cdot p = m \cdot p_A \cdot a = f_A \cdot a = f. \tag{5.2}$$

We are ready to show that $\ker a \cup \ker b$ is a kernel of p. By the definition of p, the projections of $\ker a \cup \ker b$ are made equal by p. For the universal property, let $d_1\colon D' \to D$, $d_2\colon D' \to D$ such that $p \cdot d_1 = p \cdot d_2$. Then we have

$$f \cdot d_1 \overset{(5.2)}{=} m \cdot p \cdot d_1 = m \cdot p \cdot d_2 \overset{(5.2)}{=} f \cdot d_2.$$

This means that $d_1, d_2\colon D' \rightrightarrows D$ is a competing cone w.r.t. the kernel of f. We thus obtain a unique cone morphism $u\colon D' \to \ker a \cup \ker b$ as desired.

(2) $\Rightarrow$ (3) Take $x, y, z \in Y$ with $a(x) = a(y)$ and $b(y) = b(z)$. Then $a(x)$ and $b(z)$ are identified in the pushout P:

$$p(x) = p_A \cdot a(x) = p_A \cdot a(y) = p_B \cdot b(y) = p_B \cdot b(z) = p(z).$$

This shows that (x, z) lies in $\ker a \cup \ker b$, hence we have that $a(x) = a(z)$ or $b(x) = b(z)$.

(3) $\Rightarrow$ (4) For a given $y \in Y$, there is nothing to show in the case where $[y]_a \subseteq [y]_b$. Otherwise if $[y]_a \nsubseteq [y]_b$, then there is some $x \in [y]_a$ that does not lie in $[y]_b$, i.e. such that $a(x) = a(y)$ and $b(x) \neq b(y)$. Now let $z \in [y]_b$, i.e. $b(y) = b(z)$. Then, by assumption, $a(x) = a(y) = a(z)$ or $b(x) = b(y) = b(z)$. Since the latter does not hold, we have $a(y) = a(z)$, i.e. $z \in [y]_a$. □

Example 5.16. All select routines in Example 4.8 respect compound blocks. To see this, let $k = \mathsf{select}\big(X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z\big)$.

(1) For $S \in Y$ and $[S]_z \subseteq Y$, $k := \chi^{[S]_z}_{\{S\}}\colon Y \to 3$ respects compound blocks using Lemma 5.15(4). Indeed, one proceeds by case distinction on $B \in Y$:

(a) If $B = S$, then $[B]_k = \{S\} \subseteq [S]_z = [B]_z$.

(b) If $B \in [S]_z$ and $B \neq S$, then $k(B) = 1$ so that

$$[B]_k = [S]_z \setminus \{S\} \subseteq [S]_z = [B]_z.$$

(c) Finally, if $B \in Y \setminus [S]_z$, then we have $z(B) \neq z(S)$ and therefore $[B]_z \subseteq Y \setminus [S]_z = [B]_k$, where the latter equation follows from $k(B) = 0$.

(2) The select routine returning the identity id_Y respects compound blocks, because for any morphism $z\colon Y \to Z$, $\ker \mathsf{id}_Y \cup \ker z = \ker z$ is a kernel.

(3) The constant $k = !$ respects compound blocks, because for all $B \in Y$: $[B]_z \subseteq Y = [B]_!$.

For every pair $a\colon Y \to A$, $b\colon Y \to B$ of maps, the kernel of $\langle a, b\rangle\colon Y \to A \times B$ is the intersection $\ker a \cap \ker b$. If the union $\ker a \cup \ker b$ is an equivalence relation, then every block in the partition $Y/\ker\langle a, b\rangle$ is either is an equivalence class from $Y/\ker a$ or from $X/\ker b$. That this happens can be visually illustrated as follows (see Figure 5a): Every equivalence class of $\ker a \cap \ker b$ already appears in $\ker a$ or in $\ker b$ (or both), i.e. for all $x \in Y$, we have $[x]_a = [x]_{\langle a,b\rangle}$ or $[x]_b = [x]_{\langle a,b\rangle}$. However for Figure 5b, $\{y_1, y_2, y_3\} = [y_3]_b \neq [y_3]_{\langle a,b\rangle} \neq [y_3]_a = \{y_3, y_4, y_5\}$. In the following we prove formally that whenever $\ker a \cup \ker b$ is an equivalence relation, then every equivalence class of $\ker\langle a, b\rangle$ comes from one of $\ker a$ (i.e. is in the set A') or of $\ker b$ (i.e. is in the set B').

Lemma 5.17. *Let $a\colon Y \to A$, $b\colon Y \to B$ such that $\ker a \cup \ker b$ is a kernel. Then there exist sets A', B', and maps m, q, f_A, and f_B such that the following diagrams commute:*

$$\begin{array}{ccccc} & & \overset{\langle a,b\rangle}{\frown} & & \\ Y & \xrightarrow{q} & A' + B' & \xrightarrow{m} & A \times B \end{array} \qquad \begin{array}{ccccc} A' + \bar{A}' & \xleftarrow{A'+f_A} & A' + B' & \xrightarrow{f_B+B'} & \bar{B}' + B' \\ \| & & \downarrow m & & \| \\ A & \xleftarrow{\pi_1} & A \times B & \xrightarrow{\pi_2} & B \end{array}$$

Proof. Define the sets

$$Y_A = \{x \in Y \mid [x]_a \subseteq [x]_b\}, \qquad Y_B = \{x \in Y \mid [x]_a \not\supseteq [x]_b\},$$
$$A' = \{a(x) \mid x \in Y_A\}, \qquad B' = \{b(x) \mid x \in Y_B\}.$$

By Lemma 5.15, $Y = Y_A + Y_B$. Next define $q = a' + b'\colon Y \cong Y_A + Y_B \to A' + B'$, where a' and b' are the obvious restrictions of a and b, respectively. Now put $\bar{A}' := A \setminus A'$, $\bar{B}' := B \setminus B'$, and define $f_A\colon B' \to \bar{A}'$ and $f_B\colon A' \to \bar{B}'$ by

$$f_A(b(x)) = a(x) \qquad f_B(a(x)) = b(x).$$

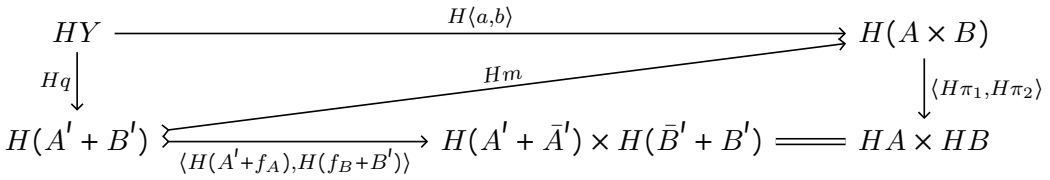
These functions are well-defined by the definition of Y_A and Y_B. Moreover, the codomain of f_A restricts to $\bar{A}'$, because $a(x) \in A'$ implies that $[x]_a \subseteq [x]_b$, contradicting $x \in Y_B$. Analogously, the codomain of f_B restricts to $\bar{B}'$. Let m be the unique map such that the right-hand diagram above commutes, i.e. m is induced by the universal property of the product $A \times B$. Since $(\mathsf{id}_{A'} + f_A) \cdot q = a$ and $(f_B + \mathsf{id}_{B'}) \cdot q = b$, we see that the left-hand diagram above commutes. □

This factorization is the main ingredient making H essentially commute with $\langle -, - \rangle$:

Proposition 5.18. *Let $a\colon Y \to A$, $b\colon Y \to B$ such that $\ker a \cup \ker b$ is a kernel, and let $H\colon \mathsf{Set} \to \mathsf{Set}$ be a zippable functor. Then we have*

$$\ker\langle Ha, Hb\rangle = \ker H\langle a, b\rangle. \tag{5.3}$$

Proof. Using the additional data provided by Lemma 5.17, we note that the following diagram commutes:



By Lemma 5.2, the composition at the bottom is a mono, because H is zippable. Hence, Hm is a mono as well. We conclude

$$\mathsf{ker}\langle Ha, Hb\rangle = \mathsf{ker}\left(\langle H\pi_1, H\pi_2\rangle \cdot H\langle a,b\rangle\right) = \mathsf{ker}(Hq) = \mathsf{ker}\, H\langle a,b\rangle,$$

using Remark 2.4(2) in the last two equalities. □

Remark 5.19. Note that Proposition 5.18 holds more generally for every functor $H\colon \mathsf{Set} \to \mathcal{D}$, where $\mathcal{D}$ is a finitely complete category.

For a select routine respecting compound blocks, we can now apply Proposition 5.18 to prove the equivalence of (5.1) and Step (3) of Algorithm 4.9:

Theorem 5.20. *If $H\colon \mathsf{Set} \to \mathsf{Set}$ is zippable and* select *respects compound blocks, then the optimization* (5.1) *is correct.*

Proof. Correctness of (5.1) means that

$$P_{i+1} = P'_{i+1} = \mathsf{ker}(H\langle \bar{q}_i, q_{i+1}\rangle \cdot \xi) = P_i \cap \mathsf{ker}(Hq_{i+1}\cdot\xi).$$

Indeed, suppose that H is zippable, and let $k = \mathsf{select}(\kappa_{P_i}, f_i)$, where $f_i\colon X/P_i \twoheadrightarrow X/Q_i$ witnesses that P_i is finer than Q_i (see Algorithm 4.9). Then we have

$$q_{i+1} = k\cdot\kappa_{P_i} \qquad \text{and} \qquad \bar{q}_i = m\cdot\kappa_{Q_i} = m\cdot f_i\cdot\kappa_{P_i}, \tag{5.4}$$

where m is obtained by the image factorization of $\bar{q}_i$. By Remark 2.4(2) we have $\mathsf{ker}\, f_i = \mathsf{ker}(m\cdot f_i)$. Since select respects compound blocks we know that $\mathsf{ker}\, f_i \cup \mathsf{ker}\, k$ is a kernel, thus so is $\mathsf{ker}(m\cdot f_i)\cup\mathsf{ker}\, k$. By Proposition 5.18, we obtain

$$\mathsf{ker}\langle H(m\cdot f_i), Hk\rangle = \mathsf{ker}\, H\langle m\cdot f_i, k\rangle,$$

which, using Remark 2.4(5), implies

$$\mathsf{ker}(\langle H(m\cdot f_i), Hk\rangle \cdot H\kappa_{P_i}\cdot\xi) = \mathsf{ker}(H\langle m\cdot f_i, k\rangle\cdot H\kappa_{P_i}\cdot\xi). \tag{5.5}$$

Thus we obtain the desired result:

$$\begin{aligned}
P_i \cap \mathsf{ker}(Hq_{i+1}\cdot\xi) &= \mathsf{ker}\left(\langle H\bar{q}_i\cdot\xi, Hq_{i+1}\cdot\xi\rangle\right) && \text{def. of } P_i\\
&= \mathsf{ker}\left(\langle H(m\cdot f_i), Hk\rangle\cdot H\kappa_{P_i}\cdot\xi\right) && \text{by (5.4)}\\
&= \mathsf{ker}(H\langle m\cdot f_i, k\rangle\cdot H\kappa_{P_i}\cdot\xi) && \text{by (5.5)}\\
&= \mathsf{ker}\left(H\langle \bar{q}_i, q_{i+1}\rangle\cdot\xi\right) && \text{by (5.4)}\\
&= \mathsf{ker}\left(H\bar{q}_{i+1}\cdot\xi\right) && \text{def. of } \bar{q}_{i+1}\\
&= P_{i+1} && \text{def. of } P_{i+1}.
\end{aligned}$$

□

Combining Theorem 5.20 and Theorem 4.20 we obtain:

Corollary 5.21. *Suppose that H is a zippable endofunctor and that* select *respects compound blocks and is progressing. Then Algorithm 4.9 with optimization* (5.1) *terminates and computes the simple quotient of a given finite H-coalgebra.*

Remark 5.22. Note that all results in this section can be formulated and proved in a Boolean topos $\mathcal{C}$ in lieu of Set, e.g. the category of nominal sets and equivariant maps. In particular, the set-theoretic statements in Lemma 5.15 and Lemma 5.17 can be formulated in the internal language of a Boolean topos, i.e. the ordinary set theory ZF (with bounded quantifiers only) without the axiom of choice, but with the law of excluded middle. A detailed definition and discussion of this language can be found, e.g. in Mac Lane and Moerdijk [MM92].

6. Refinement Interfaces

In Algorithm 4.9, we left unspecified how the kernels and partitions can be computed efficiently. In this section we introduce the notion of a *refinement interface* for the given type functor. This interface is aimed at efficient computation of the partition X/P in Algorithm 4.9, both in the initialization and in the *refinement step*, i.e. in the optimization (5.1) of Step (3) in Algorithm 4.9. In Section 7 we will assume that the type functor H comes equipped with a refinement interface which is used as a parameter for a generic initialization procedure and an implementation of (5.1) such that partition refinement on a coalgebra $\xi\colon X \to HX$ with $n = |X|$ states and m edges runs in $\mathcal{O}((m+n)\cdot \log n)$. In order to phrase and actually achieve this complexity bound, we need a notion of edges in coalgebras. This notion will also provide a representation of coalgebras, for purposes of using them as inputs to the partition refinement algorithm:

Definition 6.1. An *encoding* of a functor H consists of a set A of *labels* and a family of maps $\mathfrak{b}\colon HX \to \mathcal{B}(A\times X)$, one for every set X. The *encoding* of an H-coalgebra $\xi\colon X \to HX$ is given by the map $\langle H!, \mathfrak{b}\rangle \cdot \xi\colon X \to H1 \times \mathcal{B}(A \times X)$ and we say that the coalgebra has $n = |X|$ states and $m = \sum_{x\in X} |\mathfrak{b}(\xi(x))|$ edges.

The purpose of these functions is to enable a representation of an input coalgebra $\xi\colon X \to HX$ as a labelled graph. In concrete examples, an encoding expresses how one intuitively visualizes coalgebras (see e.g. Figure 1, Example 5.11, and Figure 3). The reader should note here that standard partition refinement for graphs on the encoding does *not* correctly minimize the input coalgebra, that is, an encoding does by no means provide a reduction of the minimization problem from coalgebras to graphs (Figure 1 illustrates how Markov chain lumping and bisimilarity in transition systems lead to different partitions). Technically speaking, behavioural equivalence in H-coalgebras is not the same as behavioural equivalence in $H1 \times \mathcal{B}(A \times (-))$-coalgebras. Moreover, it is not even assumed that the map $\mathfrak{b} : HX \to \mathcal{B}(A \times X)$ in an encoding is natural in X.

Example 6.2. (1) For $H = \mathcal{P}_\mathrm{f}$, the labels are a singleton set $A = 1$ and we define $\mathfrak{b}\colon \mathcal{P}_\mathrm{f}X \to \mathcal{B}(1 \times X) \cong \mathcal{B}(X)$ to be the obvious inclusion $\mathfrak{b}(t)(x) = 1$ if $x \in t$ and $\mathfrak{b}(t)(x) = 0$ if $x \notin t$.

(2) For $HX = \mathbb{R}^{(X)}$, the labels are non-zero real numbers $A = \mathbb{R}_{\neq 0}$, and $\mathfrak{b}\colon \mathbb{R}^{(X)} \to \mathcal{B}(\mathbb{R}_{\neq 0} \times X)$ is defined as $\mathfrak{b}(t)(r, x) = 1$ if $t(x) = r \neq 0$ and $\mathfrak{b}(t)(r, x) = 0$ if $t(x) \neq r$ or $t(x) = 0$. An example of this encoding is visualized in Figure 1b.

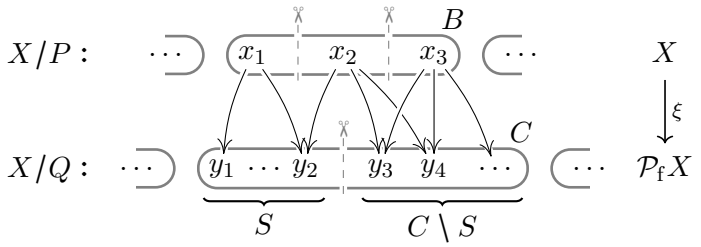


FIGURE 6. The refinement step for $\xi\colon X \to \mathcal{P}_{\mathrm{f}}X$, for $S \subseteq C \in X/Q$.

(3) For a polynomial functor H_Σ (see Example 2.6(6)), we have an encoding where the label set $A = \mathbb{N}$ describes the ordering of the components: $\flat\colon H_\Sigma X \to \mathcal{B}(\mathbb{N} \times X)$ is defined by $\flat(\sigma(x_1, \dots, x_n)) = \{(1, x_1), \dots, (n, x_n)\}$ for an n-ary symbol $\sigma \in \Sigma$. Note that the set $H_\Sigma 1$ may be identified with the signature Σ so $H! \cdot \xi$ assigns to each state x of a H_Σ-coalgebra its output symbol, and the bag $\flat \cdot \xi(x)$ contains the successor states of x together with their corresponding input $i \in \{1, \dots, n\}$.

Having a coalgebra encoded in terms of labelled edges, the following computational task is solved in the refinement step: Given a coalgebra $\xi\colon X \to HX$ as its encoding $X \to H1 \times \mathcal{B}(A \times X)$ and partitions X/P and X/Q such that P is stable w.r.t. Q (see Definition 4.3), how does splitting a block $C \in X/Q$ into $S \subseteq C$ and $C \setminus S$ affect the partition X/P? The explicit case for $H = \mathcal{P}_{\mathrm{f}}$ is visualized in Figure 6 and is computed as follows:

Example 6.3. For the finite powerset functor, fix a coalgebra $\xi\colon X \to \mathcal{P}_{\mathrm{f}}X$. We say that there is an edge from x to a subset $S \subseteq X$ if x has a successor node in S.

In the Paige-Tarjan algorithm [PT87], when a block $C \in X/Q$ is split into smaller blocks $S \subseteq C$ and $C \setminus S$, every block of $B \in X/P$ is split into (at most) three blocks as visualized in Figure 6:

(1) States $x \in B$ with edges to S but not to $C \setminus S$.

(2) States $x \in B$ with edges to both S and $C \setminus S$.

(3) States $x \in B$ with edges to $C \setminus S$ but not to S.

It is one of the key points in the analysis of the overall time complexity that this split is performed in linear time in the number of ingoing edges into S, regardless of the sizes of $C \setminus S$ and B. Roughly, this works as follows. All edges from a state $x \in X$ to the same block $C \in X/Q$ share an integer counter variable that stores how many such edges exist. Sharing a variable means that every edge is equipped with a pointer to a memory cell holding the actual counter. For example, in Figure 6, $x_2 \to y_2$, $x_2 \to y_4$, and $x_2 \to y_3$ share a variable with the value 3. When splitting C into S and $C \setminus S$, we count the edges ending in S and then compare the result with the counter variable of each of those edges, which allows us to implement the above three-way split by iterating over the source nodes $x \in B$ with edges to S:

(1) If the counters match, then all edges from x to C go indeed to S, and we move all these x to a new block.

(2) If the values mismatch, then some edges from x to C go to S and some to $C \setminus S$. Hence, we move all these x to a new block and then update the counter variables. To this end, we count the number n of edges from x to S. We re-use the existing counter variable for edges $x \to C$ and make it the variable for $x \to C \setminus S$ by decrementing it by n. For example,

in Figure 6 the counter for the edge $x_2 \to y_2$ is decremented from 3 to 2, and this counter variable is shared with $x_2 \to y_3$ and $x_2 \to y_4$, so the count for these edges is correctly decremented as well. We then allocate a new counter with value n and change the pointer of all edges $x \to S$ to this new counter. In the example in Figure 6, the pointer of the edge $x_2 \to y_2$ now points to this new counter, which holds the value 1.

(3) All the remaining $x \in B$ have no edge to S, and nothing needs to be done for them, because the counters for edges from x need not change.

We thus have refined the partition X/P such that it becomes (again) stable w.r.t. the new partition X/Q where C is split into S and $C \setminus S$.

Note that in Example 6.3, the block B is split into smaller blocks in such a way that elements x, x' end up in the same block iff $\mathcal{P}_\mathrm{f}\chi_S^C(\xi(x)) = \mathcal{P}_\mathrm{f}\chi_S^C(\xi(x))$. This split is done efficiently by maintaining an integer variable for counting edges. The present section generalizes this procedure from $\mathcal{P}_\mathrm{f}$ to a Set-functor H that implements a so-called *refinement interface*, in which the integer counter generalizes to a set W "weights" and the comparison of counters generalizes to an update function:

Definition 6.4. A *refinement interface* for a Set-functor H equipped with a functor encoding is formed by a set W of *weights* and functions

$$\mathsf{init}\colon H1 \times \mathcal{B}_\mathrm{f}A \to W, \qquad \mathsf{update}\colon \mathcal{B}_\mathrm{f}A \times W \to W \times H3 \times W$$

such that there exists a family of *weight maps* $w\colon \mathcal{P}_\mathrm{f}X \to (HX \to W)$ such that for all $S \subseteq C \subseteq X$, the diagrams

$$\begin{array}{ccc} H1 \times \mathcal{B}_\mathrm{f}A & \xrightarrow{\mathsf{init}} & W \\ {\scriptstyle \langle H!, \mathsf{fil}_X \cdot \flat \rangle}\uparrow & \nearrow_{w(X)} & \\ HX & & \end{array} \qquad \begin{array}{ccc} \mathcal{B}_\mathrm{f}A \times W & \xrightarrow{\mathsf{update}} & W \times H3 \times W \\ {\scriptstyle \langle \mathsf{fil}_S \cdot \flat, w(C) \rangle}\uparrow & \nearrow_{\langle w(S), H\chi_S^C, w(C\setminus S)\rangle} & \\ HX & & \end{array} \tag{6.1}$$

commute, where $\mathsf{fil}_S\colon \mathcal{B}_\mathrm{f}(A \times X) \to \mathcal{B}_\mathrm{f}(A)$ is the filter function $\mathsf{fil}_S(f)(a) = \sum_{y \in S} f(a, y)$ for $S \subseteq X$ (e.g. $\mathsf{fil}_X = \mathcal{B}_\mathrm{f}\pi_1$).

Remark 6.5. (1) Note that for a coalgebra $\xi\colon X \to HX$, $\mathsf{fil}_S \cdot \flat \cdot \xi(x)$ is the bag of labels of all edges from x to $S \subseteq X$ in the encoding of ξ.

(2) Observe that fil_S is natural in A, i.e. for every map $h\colon A \to A'$ the following square commutes:

$$\begin{array}{ccc} \mathcal{B}_\mathrm{f}(A \times X) & \xrightarrow{\mathsf{fil}_S} & \mathcal{B}_\mathrm{f}A \\ {\scriptstyle \mathcal{B}_\mathrm{f}(h \times X)}\downarrow & & \downarrow{\scriptstyle \mathcal{B}_\mathrm{f}h} \\ \mathcal{B}_\mathrm{f}(A' \times X) & \xrightarrow{\mathsf{fil}_S} & \mathcal{B}_\mathrm{f}A'. \end{array}$$

Indeed, we have for all $f \in \mathcal{B}_{\mathrm{f}}(A \times X)$

$$\begin{aligned}
\mathsf{fil}_S \cdot \mathcal{B}_{\mathrm{f}}(h \times X)(f) &= \mathsf{fil}_S\big((a', x) \mapsto \sum_{(h\times X)(a,y)=(a',x)} f(a,y)\big) \\
&= \mathsf{fil}_S\big((a', x) \mapsto \sum_{h(a)=a'} f(a,x)\big) \\
&= a' \mapsto \sum_{y\in S} \sum_{h(a)=a'} f(a,y) \\
&= a' \mapsto \sum_{h(a)=a'} \sum_{y\in S} f(a,y) \\
&= \mathcal{B}_{\mathrm{f}} h\big(a \mapsto \sum_{y\in S} f(a,y)\big) \\
&= \mathcal{B}_{\mathrm{f}} h \cdot \mathsf{fil}_S(f).
\end{aligned}$$

Informally, for a given coalgebra $\xi\colon X \to HX$, the above axioms for init and update can be understood as a kind of contract that their implementation for a given functor needs to fulfil: init receives in its first argument the information which states of X are (non-)terminating, in its second argument the bag of labels of all outgoing edges of a state $x \in X$ in the graph representation of ξ, and it returns the total weight of those edges. The operation update receives a pair consisting of the bag of labels of all edges from some state $x \in X$ into the set $S \subseteq X$ and the weight of all edges from x to $C \subseteq X$, and from only this information (in particular update does not know S and C explicitly) it computes the triple consisting of the weight of edges from x to S, the result of $H\chi_S^C \cdot \xi(x)$ and the weight of edges from x to $C \setminus S$ (in Example 6.3, the number of edges from x to S, the value for the three way split, and the number of edges from x to $C \setminus S$). The significance of the set $H3$ is that when using a set $S \subseteq C \subseteq X$ as a splitter, we want to split every block B in such a way that it becomes *compatible* with S and $C \setminus S$, i.e. we group the elements $x \in B$ by the value of $H\chi_S^C \cdot \xi(x) \in H3$. The set W depends on the functor. But in most cases $W = H2$ and $w(C) = H\chi_C\colon HX \to H2$ are sufficient.

In implementations, the refinement interface does not need to provide w explicitly, because the algorithm will compute the values of w incrementally using (6.1), and $\flat$ need not be implemented because we assume the input coalgebra to be already *encoded* via $\flat$.

Example 6.6. The refinement interface for the powerset functor $H = \mathcal{P}_{\mathrm{f}}$ needs to count the edges into blocks C, so that we know in the refinement with $S \subseteq C$ whether there are edges to $C \setminus S$, as in Example 6.3. For a natural number n, we define the auxiliary function

$$(-) \overset{?}{>} 0\colon \mathbb{N} \to 2 \qquad \text{by } (n \overset{?}{>} 0) = \min(n, 1) = \begin{cases} 1 & \text{if } n > 0 \\ 0 & \text{else.} \end{cases}$$

For the refinement interface we use the following weights and labels:

$$A = 1, \quad W = 2 \times \mathbb{N}, \quad w\colon \mathcal{P}_{\mathrm{f}}X \to (\mathcal{P}_{\mathrm{f}}X \to 2 \times \mathbb{N}) \text{ with } w(C)(t) = (|t \setminus C| \overset{?}{>} 0, |t \cap C|).$$

For a map $\xi\colon X \to \mathcal{P}_{\mathrm{f}}X$, a state $x \in X$, and a block $C \subseteq X$, the weight $w(C)(\xi(x))$ is a tuple whose first component tells us whether there is any edge from x to $X \setminus C$ and whose second component is the number of edges from x to C. Since we have a singleton label alphabet $A = 1$, every bag of labels is just a natural number, because $\mathcal{B}_{\mathrm{f}}A = \mathcal{B}_{\mathrm{f}}1 \cong \mathbb{N}$. The

functions

$$\mathsf{init}\colon \mathcal{P}_\mathrm{f}1 \times \mathbb{N} \to 2 \times \mathbb{N} \qquad \text{and} \qquad \mathsf{update}\colon \mathbb{N} \times (2 \times \mathbb{N}) \to (2 \times \mathbb{N}) \times \mathcal{P}_\mathrm{f}3 \times (2 \times \mathbb{N})$$

in the refinement interface for $\mathcal{P}_\mathrm{f}$ are implemented as follows:

$$\mathsf{init}(z, n) = (0, n) \qquad \text{and} \qquad \mathsf{update}(n_S, (r, n_C)) = \big((r \vee (n_{C\setminus S} \overset{?}{>} 0), n_S), (r, n_{C\setminus S} \overset{?}{>} 0, n_S \overset{?}{>} 0), (r \vee (n_S \overset{?}{>} 0), n_{C\setminus S})\big),$$

where $n_{C\setminus S} := \max(n_C - n_S, 0)$, $\vee\colon 2 \times 2 \to 2$ is disjunction, and the middle return value in $\mathcal{P}_\mathrm{f}3$ is written as a bit vector of length three. The axioms in (6.1) ensure that $n_S, n_C, n_{C\setminus S}$ can be understood as the numbers of edges to S, C, and $C \setminus S$, respectively.

For the verification of (6.1), note that for all $S \subseteq X$, the map $\mathsf{fil}_S \cdot \flat\colon \mathcal{P}_\mathrm{f}X \to \mathbb{N}$ is given by $\mathsf{fil}_S(\flat(t)) = |t \cap S|$. Hence, we see that for all $t \in \mathcal{P}_\mathrm{f}X$ we have

$$\mathsf{init}(\mathcal{P}_\mathrm{f}!(t), \mathsf{fil}_X(\flat(t))) = (0, \mathsf{fil}_X(\flat(t))) = (|t \setminus X| \overset{?}{>} 0, |t \cap X|) = w(X)(t).$$

In the verification of the axiom of update, the parameters expand as follows:

$$\mathsf{update}(\mathsf{fil}_S(\flat(t)), w(C)(t)) = \mathsf{update}(|t \cap S|, (|t \setminus C| \overset{?}{>} 0, |t \cap C|)) \Rightarrow \begin{cases} n_S = |t \cap S| \\ r \;\; = |t \setminus C| \overset{?}{>} 0 \\ n_C = |t \cap C| \end{cases}$$

So $n_{C\setminus S} = \max(n_C - n_S, 0) = \max(|t \cap C| - |t \cap S|, 0) = |t \cap (C \setminus S)|$, since $S \subseteq C$. The remaining steps of the verification are performed component-wise: for $x = \mathsf{update}(\mathsf{fil}_S(\flat(t)), w(C)(t))$ we have

$$\begin{aligned}
\pi_1(x) &= \big((|t \setminus C| \overset{?}{>} 0) \vee (|t \cap (C \setminus S)| \overset{?}{>} 0), |t \cap S|\big) \\
&= \big(|t \setminus S| \overset{?}{>} 0, |t \cap S|\big) = w(S)(t), \\
\pi_2(x) &= (|t \setminus C| \overset{?}{>} 0, |t \cap (C \setminus S)| \overset{?}{>} 0, |t \cap S| \overset{?}{>} 0) \\
&= \Big(\bigvee_{\substack{y \in t \\ \chi^C_S(y)=0}} 1, \bigvee_{\substack{y \in t \\ \chi^C_S(y)=1}} 1, \bigvee_{\substack{y \in t \\ \chi^C_S(y)=2}} 1\Big) \qquad \text{as an element of } \mathcal{P}_\mathrm{f}3 \\
&= \{\chi^C_S(y) \mid y \in t\} = \mathcal{P}_\mathrm{f}\chi^C_S(t), \\
\pi_3(x) &= \big((|t \setminus C| \overset{?}{>} 0) \vee (|t \cap S| \overset{?}{>} 0), |t \cap (C \setminus S)|\big) \\
&= \big(|t \setminus (C \setminus S)| \overset{?}{>} 0, |t \cap (C \setminus S)|\big) = w(C\setminus S)(t).
\end{aligned}$$

Example 6.7. In the following examples, we always take

$$W = H2 \qquad \text{and} \qquad w(C) = H\chi_C\colon HX \to H2.$$

We also use the helper function

$$\mathsf{val} := \langle H(=2), \mathsf{id}_{H3}, H(=1)\rangle\colon H3 \to H2 \times H3 \times H2,$$

where $(=x)\colon 3 \to 2$ is the equality check for $x \in \{1, 2\}$, and in each case define

$$\mathsf{update} = (\mathcal{B}_\mathrm{f}A \times H2 \xrightarrow{\mathsf{up}} H3 \xrightarrow{\mathsf{val}} H2 \times H3 \times H2),$$

for a function $\mathsf{up}\colon \mathcal{B}_\mathrm{f}A \times H2 \to H3$ that is defined individually for every functor.

For the verification of (6.1) note that, in general, for $S \subseteq C \subseteq X$, we have

$$\mathsf{val} \cdot H\chi^C_S = \langle H\chi_S, H\chi^C_S, H\chi_{C\setminus S}\rangle.$$

Hence, to verify the axiom for $\mathsf{update} = \mathsf{val} \cdot \mathsf{up}$ it suffices to verify that

$$\mathsf{up} \cdot \langle \mathsf{fil}_S \cdot \mathsf{b}, H\chi_C \rangle = H\chi_S^C; \tag{6.2}$$

in fact, using $w(C) = H\chi_C$ we have:

$$\begin{aligned} \mathsf{update} \cdot \langle \mathsf{fil}_S \cdot \mathsf{b}, w(C) \rangle &= \mathsf{val} \cdot \mathsf{up} \cdot \langle \mathsf{fil}_S \cdot \mathsf{b}, H\chi_C \rangle \\ &= \mathsf{val} \cdot H\chi_S^C \\ &= \langle H\chi_S, H\chi_S^C, H\chi_{C\setminus S} \rangle \\ &= \langle w(S), H\chi_S^C, w(C \setminus S) \rangle. \end{aligned}$$

(1) For the monoid-valued functor $H = G^{(-)}$ over an Abelian group $(G, +, 0)$, we take labels $A = G$ and define $\mathsf{b}(f) = \{(f(y), y) \mid y \in X, f(y) \neq 0\}$ (which is finite because f is finitely supported). With $W = H2 = G \times G$, the weight $w(C) = H\chi_C \colon HX \to G \times G$ assigns to $f \in HX = G^{(X)}$ the pair of accumulated weights of $X \setminus C$ and C under f:

$$w(C)(f) = \Big(\sum_{y \in X\setminus C} f(y), \sum_{y \in C} f(y) \Big).$$

The remaining functions are

$$\mathsf{init}(h_1, e) = (0, \Sigma e) \quad \text{and} \quad \mathsf{up}(e, (r, c)) = (r, c - \Sigma e, \Sigma e),$$

where $\Sigma \colon \mathcal{B}_{\mathrm{f}} G \to G$ is the obvious summation map assigning to a bag of elements of G their sum in G.

Then for all $f \in HX = G^{(X)}$ and $S \subseteq C \subseteq X$, we have:

$$\begin{aligned} \mathsf{init}(H!(f), \mathsf{fil}_X \cdot \mathsf{b}(f)) &= \big(0, \textstyle\sum \mathcal{B}_{\mathrm{f}} \pi_1 \cdot \mathsf{b}(f)\big) \\ &= \big(0, \sum_{y \in X} f(y)\big) = G^{(\chi_X)}(f) = w(X)(f), \\ \mathsf{up}(\mathsf{fil}_S(\mathsf{b}(f)), H\chi_C(f)) &= \mathsf{up}\big(\{f(y) \mid y \in S\}, \big(\sum_{y \in X\setminus C} f(y), \sum_{y \in C} f(y)\big)\big) \\ &= \big(\sum_{y \in X\setminus C} f(y), \sum_{y \in C} f(y) - \sum_{y \in S} f(y), \sum_{y \in S} f(y)\big) \\ &\overset{S\subseteq C}{=} \big(\sum_{y \in X\setminus C} f(y), \sum_{y \in C\setminus S} f(y), \sum_{y \in S} f(y)\big) = H\chi_S^C(f), \end{aligned}$$

which verifies (6.2) and therefore (6.1).

(2) As a special case, we obtain a refinement interface for the functor $\mathbb{R}^{(-)}$, and from this we can derive one for the distribution functor $\mathcal{D}$, a subfunctor of $\mathbb{R}^{(-)}_{\geq 0}$, the following init and up functions:

$$\mathsf{init}(h_1, e) = (0, 1) \in \mathcal{D}2 \subset [0, 1]^2 \qquad \text{and} \qquad \mathsf{up}(e, (r, c)) = (r, c - \Sigma e, \Sigma e),$$

if the latter lies in $\mathcal{D}3$, and $\mathsf{up}(e, (r, c)) = (0, 0, 1)$ otherwise.

The axiom for init clearly holds since for every $f \in \mathcal{D}X$, we have $\Sigma \mathcal{B}\pi_1 \cdot \mathsf{b}(f) = \sum_{y \in X} f(y) = 1$.

The axiom (6.2) for up is proved as in the previous example; in fact, note that for an $f \in \mathcal{D}X$ all components of the triple $\big(\sum_{y\in X\setminus C} f(y), \sum_{y\in C\setminus S} f(y), \sum_{y\in S} f(y)\big)$ are in $[0,1]$ and their sum is $\sum_{y\in X} f(y) = 1$. Thus, this triple lies in $\mathcal{D}3$ and is equal to $\mathcal{D}\chi_S^C(f)$.

(3) Similarly, one obtains a refinement interface for $\mathcal{B}_\mathrm{f} = \mathbb{N}^{(-)}$, adjusting the one for $\mathbb{Z}^{(-)}$; in fact, init remains unchanged and $\mathsf{up}(e,(r,c)) = (r, c-\Sigma e, \Sigma e)$ if the middle component is a natural number and $(0,0,0)$ otherwise.

To verify (6.1) for the refinement interface for $\mathbb{N}^{(-)}$ we argue similarly as in point (2) above: If f lies in $\mathbb{N}^{(X)} \subseteq \mathbb{Z}^{(X)}$, i.e. has only non-negative components, then only non-negative components appear in the data returned by up and update, so up and update restrict from $\mathbb{Z}^{(-)}$ to $\mathbb{N}^{(-)}$ as desired.

(4) Given a polynomial functor H_Σ for the signature Σ, recall from Example 6.2 that the labels $A = \mathbb{N}$ records the generators contained in a shallow term together with their indices:

$$\flat(\sigma(y_1,\dots,y_n)) = \{(1,y_1),\dots,(n,y_n)\}.$$

Then the functor interface is given by $w(C) = H\chi_C$ and

$$\begin{aligned}\mathsf{init}(\sigma(0,\dots,0), f) &= \sigma(1,\dots,1)\\ \mathsf{up}(I, \sigma(b_1,\dots,b_n)) &= \sigma(b_1 + (1\in I),\dots,b_i + (i\in I),\dots,b_n + (n\in I)).\end{aligned}$$

Here $b_i + (i \in I)$ means $b_i + 1$ if $i \in I$ and b_i otherwise.

To verify (6.1), let $S \subseteq C \subseteq X$, $t = \sigma(y_1,\dots,y_n) \in H_\Sigma X$ with σ of arity n, let $b_i = \chi_C(y_i)$, and $I = \{1 \le i \le n \mid y_i \in S\}$. Then we have:

$$\begin{aligned}\mathsf{init}(H_\Sigma !(t), \mathcal{B}_\mathrm{f}\pi_1 \cdot \flat(t)) &= \mathsf{init}(\sigma(0,\dots,0), \mathcal{B}_\mathrm{f}\pi_1(\{(1,y_1),\dots,(n,y_n)\}))\\ &= \mathsf{init}(\sigma(0,\dots,0), \{1,\dots,n\}) = \sigma(1,\dots,1)\\ &= \sigma(\chi_X(y_1),\dots,\chi_X(y_n)) = H_\Sigma\chi_X(t).\\ \mathsf{up}(\mathsf{fil}_S \cdot \flat(t), H_\Sigma\chi_C(t)) &= \mathsf{up}(\mathsf{fil}_S(\{(1,y_1),\dots,(n,y_n)\}), \sigma(b_1,\dots,b_n))\\ &= \mathsf{up}(I, \sigma(b_1,\dots,b_n))\\ &= \sigma(b_1 + (1\in I),\dots,b_i + (i\in I),\dots,b_n + (n\in I))\\ &= \sigma(\chi_S^C(y_1),\dots,\chi_S^C(y_i),\dots,\chi_S^C(y_n))\\ &= H_\Sigma\chi_S^C(t).\end{aligned}$$

In the second last step we use that:

$$\begin{aligned} y_i \in X\setminus C &\Rightarrow b_i + (i\in I) = 0+0 = 0 = \chi_S^C(y_i),\\ y_i \in C\setminus S &\Rightarrow b_i + (i\in I) = 1+0 = 1 = \chi_S^C(y_i),\\ y_i \in S &\Rightarrow b_i + (i\in I) = 1+1 = 2 = \chi_S^C(y_i).\end{aligned}$$

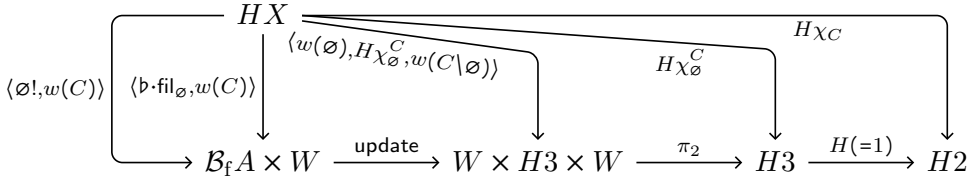
By Proposition 2.13, the interface for Abeliean-group-valued functors can also be used for monoid-valued functors $M^{(-)}$ for cancellative monoids M (as these embed into Abelian groups). In further work [DMSW19], we provide a refinement interface for monoid-valued functors $M^{(-)}$ over unrestricted monoids M; this also yields a refinement interface for $\mathcal{P}_\mathrm{f}$ as the monoid-valued functor $(2,\vee,0)^{(-)}$. However, the more general refinement interface is

less efficient than the specific interface for $\mathcal{P}_\mathrm{f}$ described in Example 6.6; in particular it does not yield the linear run-time complexity we seek here and require next in Assumption 6.9.

The next result shows that for every refinement interface the weight function $w(C)$ provides at least as much information as $H\chi_C$.

Proposition 6.8. *For every refinement interface,* $H\chi_C = H(=1) \cdot \pi_2 \cdot \mathsf{update}(\varnothing) \cdot w(C)$.

Proof. The axiom for update and definition of $\mathsf{fil}_\varnothing$ makes the following diagram commute:



□

Assumption 6.9. From now on, we assume that $H\colon \mathsf{Set} \to \mathsf{Set}$ is zippable and given together with a refinement interface such that init and update run in linear time, $H3$ is linearly ordered, and its elements can be compared in constant time.

Remark 6.10. We implicitly impose some standard assumptions regarding arithmetic on our computational model, namely that integers can be stored in atomic memory cells and the usual operations on them, e.g. addition and comparison, run in constant time.

Example 6.11. The refinement interfaces in Examples 6.6 and 6.7 satisfy Assumption 6.9:

(1) For all examples using the val-function from Example 6.7, first note that val runs in linear time (with a constant factor of 3, because val essentially returns three copies of its input). Hence update runs in linear time if up does.

(2) For monoid-valued functors $G^{(-)}$ over an abelian group G, for $\mathbb{N}^{(-)}$ and for $\mathcal{D}$, all the operations, including the summation Σe (see Example 6.7.(1)), run in time linear in the size of the input. If the elements $g \in G$ have a bounded representation, i.e. fit into boundedly many memory cells, then so do the elements of $G^{(2)}$; thus comparing elements of $g_1, g_2 \in G^{(2)}$ can be performed in constant time. (By our global assumptions as per Remark 6.10, this includes cases where G consists of integer or rational numbers.)

(3) Given a polynomial functor H_Σ, we assume that operation symbols $\sigma \in \Sigma$ are encoded as integers. As per Remark 6.10, we can then assume that operation symbols can be compared in constant time. If the signature has *bounded arities* (i.e. there is a finite bound on the arity of all symbols in Σ), then the maximum arity of operation symbols present in a given coalgebra $\xi : X \to H_\Sigma X$ is bounded independently of ξ, so the comparison of the generators in two shallow Σ-terms $t_1, t_2 \in H_\Sigma 3$ runs in constant time as well.

(a) The first parameter of init is of type $H_\Sigma 1$ and can be encoded by an operation symbol σ, i.e. by an integer. Let $t \in H_\Sigma 2$ be fixed. Then we explicitly implement

$$\mathsf{init}(\sigma, f) = \begin{cases} \sigma(\overbrace{1, \dots, 1}^{\text{arity } \sigma \text{ many}}) & \text{if } \mathrm{arity}(\sigma) = |f| \\ t & \text{otherwise.} \end{cases}$$

Both the check and the construction of $\sigma(1, \dots, 1)$ run in linear time in the size of $f \in \mathcal{B}_\mathrm{f}\mathbb{N}$, or in constant time in the second case, since t was fixed beforehand.

(b) In $\mathsf{up}(I, \sigma(b_1, \ldots, b_n))$, we cannot naively check all the $1 \in I, \ldots, n \in I$ queries, since this would lead to quadratic run-time. Instead, we precompute all results of possible queries:

1: Define an array elem with indices $1 \ldots n$, where each cell stores a value in 2.
2: Initialize elem to 0 everywhere.
3: **for** $i \in I$ with $i \leq n$ **do** $\mathsf{elem}[i] := 1$.
4: **return** $\sigma(b_1 + \mathsf{elem}[1], \ldots, b_n + \mathsf{elem}[n])$.

The running time of every line is bound by $|I| + n$.

7. Efficient Partition Refinement

We now proceed to present Algorithm 4.9 in a concrete form that is parametric in a refinement interface for the coalgebraic type functor H. We will prove the correctness in Section 7.2 and then analyse the efficient run-time in Section 7.3.

We continue to work under Assumption 6.9.

7.1. The Concrete Algorithm.

In the actual implementation, we need to address edges explicitly, and so we define the following maps:

Definition 7.1. Given a functor H, equipped with an encoding, and a map $\xi\colon X \to HX$, the set E of *edges* is defined by

$$E := \coprod_{x \in X} \coprod_{(a,y) \in A \times X} (\flat \cdot \xi(x))(a, y)$$

where $(\flat \cdot \xi(x))(a, y) \in \mathbb{N}$ is considered as a finite ordinal number. The *encoding* of ξ is represented by two functions (implemented as arrays):

$$\mathsf{type} = \left(X \xrightarrow{\xi} HX \xrightarrow{H!} H1\right) \quad \text{and} \quad \mathsf{graph}\colon E \longrightarrow X \times A \times X \text{ with } \mathsf{in}_x(\mathsf{in}_{a,y} n) \mapsto (x, a, y).$$

Remark 7.2. For the complexity result, we assume that the partitions X/P and X/Q are implemented in such a way that we can add and remove elements in constant time, remove and create blocks in constant time, and find the surrounding block of an element $x \in X$ or $y \in X$ in constant time.

(1) One possible implementation is by doubly linked lists of the blocks, where each block is in turn encoded as a doubly linked list of its elements, and additionally every element $x \in X$, $y \in X$ holds a pointer to the corresponding list entry in the blocks containing them [PT87].

(2) An alternative implementation is the *refinable partition* data structure [VF10], in which the partition is an array of elements and elements in the same block appear consecutively in the array. So a block of the partition consists of two indices, denoting the interval in the array of all elements.

Both implementations require $\mathcal{O}(n)$ space, where n is the number of elements, but the linked list approach has a much higher constant factor due to the high number of pointers. Thus, the implementation of our algorithm [DMSW19] uses the refinable partition structure.

The algorithm maintains the following mutable data structures:

- An array $\mathsf{toSub}\colon X \to \mathcal{B}_{\mathrm{f}} E$, mapping $x \in X$ to its outgoing edges ending in the currently processed subblock.

- A pointer mapping edges to memory addresses: $\mathsf{lastW}\colon E \to \mathbb{N}$. By this pointer, we achieve that edges with a common source node $x \in X$ and a common target block $C \in X/Q$ point to the same memory cell holding $w(C, \xi(x))$, as in the concrete Example 6.3 above.
- The actual store for weights $\mathsf{deref}\colon \mathbb{N} \to W$, into which lastW points.
- For each block $B \in X/P$ a set of markings $\mathsf{mark}_B \subseteq B \times \mathbb{N}$, that collects those pairs consisting of a state $x \in B$ and the pointer to $w(C, \xi(x))$ in the store deref of weights for those x that have an outgoing edge to the current subblock S during the splitting operation. Initially, mark_B is empty for every newly created block B and after each refinement step, mark_B is emptied again.

Notation 7.3. In the following we write $e = x \xrightarrow{a} y$ in lieu of $\mathsf{graph}(e) = (x, a, y)$. We also overload notation and write the weight function from the refinement interface of H in its uncurried form as $w\colon \mathcal{P}_{\mathrm{f}} X \times HX \to W$.

Definition 7.4 (Invariants)**.** Our correctness proof below establishes that the following properties hold before and after each call to our splitting routine; we call them *the invariants*:

(1) The array toSub is empty, i.e. for all $x \in X, \mathsf{toSub}(x) = \emptyset$.

(2) The pointers in lastW are the same for two edges if and only if their source nodes and target blocks agree: for every $e_1 = x_1 \xrightarrow{a_1} y_1$ and $e_2 = x_2 \xrightarrow{a_2} y_2$ we have

$$\mathsf{lastW}(e_1) = \mathsf{lastW}(e_2) \iff x_1 = x_2 \text{ and } [y_1]_{\kappa_Q} = [y_2]_{\kappa_Q}.$$

(3) The pointers in lastW point to the correct weights in the store of weights, i.e. for every $e = x \xrightarrow{a} y$ in E and $C := [y]_{\kappa_Q} \in X/Q$, we have

$$w(C, \xi(x)) = \mathsf{deref} \cdot \mathsf{lastW}(e).$$

(4) For every block $C \in X/Q$, The partition X/P is stable w.r.t. χ_C, i.e. for every $x_1, x_2 \in B \in X/P$ and $C \in X/Q$, we have $(x_1, x_2) \in \mathsf{ker}(H\chi_C \cdot \xi)$, cf. Definition 4.3.

In the following code listings, we use square brackets for array lookups and updates in order to emphasize they run in constant time. We assume that the functions $\mathsf{graph}\colon E \to X \times A \times X$ and $\mathsf{type}\colon X \to H1$ are implemented as arrays. In the initialization step, we precompute the additional static array $\mathsf{pred}\colon X \to \mathcal{P}_{\mathrm{f}} E$,

$$\mathsf{pred}(y) = \{e \in E \mid e = x \xrightarrow{a} y\}$$

which holds for every $y \in X$ the set of incoming edges.

Sets and bags are implemented as lists. We only insert elements into sets not yet containing them.

Definition 7.5. We say that we *group* (or *split*) a finite set Z by a map $f\colon Z \to Z'$ to indicate that we compute $[-]_f$, i.e. we partition Z according the values of its elements under f.

This is implemented by first sorting the elements $z \in Z$ by a binary encoding of $f(z)$ using any $\mathcal{O}(|Z| \cdot \log |Z|)$ sorting algorithm, and then grouping elements with the same $f(z)$ into blocks. In order to keep the overall complexity for the grouping operations low enough, one needs to use a possible majority candidate during sorting, following Valmari and Franceschinis [VF10]. The algorithm computing the initial partition is listed in Figure 7. The first two lines initialize $\mathsf{toSub}(x)$ to be the bag of all outgoing edges of x and compute

INITIALIZATION

1: **for** $e \in E$, $e = x \xrightarrow{a} y$ **do**
2: add e to $\mathsf{toSub}[x]$ and $\mathsf{pred}[y]$
3: **for** $x \in X$ **do**
4: $p_X :=$ new cell in deref containing $\mathsf{init}(\mathsf{type}[x], \mathcal{B}_\mathrm{f}(\pi_2 \cdot \mathsf{graph})(\mathsf{toSub}[x]))$
5: **for** $e \in \mathsf{toSub}[x]$ **do** $\mathsf{lastW}[e] = p_X$
6: $\mathsf{toSub}[x] := \varnothing$
7: $X/P :=$ group X by $\mathsf{type}\colon X \to H1$; $X/Q := \{X\}$.

FIGURE 7. The initialization procedure.

$\mathsf{pred}(y)$. Then the loop in lines 3–6 initializes the array lastW, and finally the two partitions X/P and X/Q are initialized in line 7.

Lemma 7.6. *The initialization procedure runs in time $\mathcal{O}(|E| + |X| \cdot \log |X|)$ and the result satisfies the invariants.*

Proof. The grouping in line 7 takes $\mathcal{O}(|X| \cdot \log |X|)$ time. The first loop takes $\mathcal{O}(|E|)$ steps, and the second one takes $\mathcal{O}(|X| + |E|)$ time in total over all $x \in X$ since init is assumed to run in linear time. For the invariants:

(1) By line 6.

(2) Let $e_1 = x_1 \xrightarrow{a_1} y_1$ and $e_2 = x_2 \xrightarrow{a_2} y_2$. Since $[y_1]_{\kappa_Q} = [y_2]_{\kappa_Q}$ holds for all y_1, y_2, it suffices to show that $\mathsf{lastW}(e_1) = \mathsf{lastW}(e_2)$ iff $x_1 = x_2$. This holds after the procedure because p_X in line 4 is the address of a new memory cell, whence $\mathsf{lastW}(e_1)$ and $\mathsf{lastW}(e_2)$ are equal iff they are assigned their value in the same **for** loop in line 5, equivalently, if $e_1, e_2 \in \mathsf{toSub}(x)$ for some x. Equivalently, $x_1 = x = x_2$ because after line 2, $\mathsf{toSub}(x)$ is the bag of all outgoing edges of x.

(3) First, we have for every $x \in X$ that

$$\begin{aligned}
\mathcal{B}_\mathrm{f}(\pi_2 \cdot \mathsf{graph})(\mathsf{toSub}(x)) &= \mathcal{B}_\mathrm{f}(\pi_2 \cdot \mathsf{graph})(\{e \mid e \in E, e = x \xrightarrow{a} y\} && \text{(line 2)}\\
&= \{a \mid x \xrightarrow{a} y \text{ in } E\} && \text{(Def. 7.1)}\\
&= \mathcal{B}_\mathrm{f}\pi_1\big(\{(a, y) \mid x \xrightarrow{a} y \text{ in } E\}\big)\\
&= \mathcal{B}_\mathrm{f}\pi_1 \cdot \flat \cdot \xi(x) && \text{(Def. 7.1)}
\end{aligned}$$

where the comprehensions are read as *multiset* comprehensions, i.e. multiple edges with the same label a generate multiple occurrences of a. Then we use this in the second step below to see that we have for every $e = x \xrightarrow{a} y$ in E:

$$\begin{aligned}
\mathsf{deref} \cdot \mathsf{lastW}(e) &= \mathsf{init}(\mathsf{type}(x), \mathcal{B}_\mathrm{f}(\pi_2 \cdot \mathsf{graph})(\mathsf{toSub}(x))) && \text{(lines 4 and 5)}\\
&= \mathsf{init}(\mathsf{type}(x), \mathcal{B}_\mathrm{f}\pi_1 \cdot \flat \cdot \xi(x))\\
&= \mathsf{init}(H! \cdot \xi(x), \mathcal{B}_\mathrm{f}\pi_1 \cdot \flat \cdot \xi(x)) && \text{(Def. of } \mathsf{type})\\
&= w(X, \xi(x)) && \text{by (6.1)},
\end{aligned}$$

and we are done since $X/Q = \{X\}$.

(4) Since $\mathsf{ker}(H\chi_X \cdot \xi) = \mathsf{ker}(H! \cdot \xi)$, this is just the way X/P is constructed. □

The algorithm for a single refinement step is listed in Figure 8. It receives as input the the partition X/P, where the kernel P is stable w.r.t. $\kappa_Q\colon X \twoheadrightarrow X/Q$, and a subblock $S \subseteq X$

```
Split(X/P, S ⊆ X)
 1: M := ∅ ⊆ X/P × H3
 2: for y ∈ S, e ∈ pred[y] do
 3:     x --a--> y := e
 4:     B := block with x ∈ B ∈ X/P
 5:     if mark_B is empty then
 6:         w_C^x := deref · lastW[e]
 7:         v_∅ := π2 · update(∅, w_C^x)
 8:         add (B, v_∅) to M
 9:     if toSub[x] = ∅ then
10:         add (x, lastW[e]) to mark_B
11:     add e to toSub[x]
```

(A) Collecting predecessor blocks

```
12: for (B, v_∅) ∈ M do
13:     B_≠∅ := ∅ ⊆ X × H3
14:     for (x, p_C) in mark_B do
15:         ℓ := B_f(π2 · graph)(toSub[x])
16:         (w_S^x, v^x, w_{C\S}^x) := update(ℓ, deref[p_C])
17:         deref[p_C] := w_{C\S}^x
18:         p_S := new cell containing w_S^x
19:         for e ∈ toSub[x] do lastW[e] := p_S
20:         toSub[x] := ∅
21:         if v^x ≠ v_∅ then
22:             remove x from B
23:             insert (x, v^x) into B_≠∅
24:     mark_B := ∅
25:     B_1 × {v_1}, ..., B_ℓ × {v_ℓ} :=
          group B_≠∅ by π2: X × H3 → H3
26:     insert B_1, ..., B_ℓ := into X/P
```

(B) Splitting predecessor blocks

FIGURE 8. Refining X/P for $S \subseteq C$, $C \in X/Q$, i.e. making it stable w.r.t. $\chi_S^C \colon X \to 3$.

contained in $C \in X/Q$. Its task is to split the blocks in X/P in such a way that P becomes stable w.r.t. $\langle \kappa_Q, \chi_S^C \rangle$, i.e. stable w.r.t. κ_Q after C has been split into S and C/S in X/Q.

In the first part, all blocks $B \in X/P$ that have an edge into S are collected, together with $v_\emptyset = H\chi_S^C \cdot \xi(x) \in H3$ for all $x \in B$ that have no edge into S. For each $x \in X$, $\mathsf{toSub}[x]$ collects the edges from x into S. The markings mark_B list those elements $x \in B$ that have an edge into S, together with one of the pointers in lastW that point to $w(C, x)$ in the store deref of weights.

In the second part, each block B with an edge into S is split by $H\chi_S^C \cdot \xi$, cf. Example 6.3. First, for every $(x, p_C) \in \mathsf{mark}_B$, we compute $w(S, x)$, $v^x = H\chi_S^C \cdot \xi(x)$, and $w(C \setminus S, x)$ using update. Then, the weight of all edges $x \to C \setminus S$ is updated to $w(C \setminus S, x)$ and the weight of all edges $x \to S$ is stored in a new cell containing $w(S, x)$. For all unmarked $x \in B$, we know that $H\chi_S^C \cdot \xi(x) = v_\emptyset$; so all x with $v^x = v_\emptyset$ stay in B. All other $x \in B$ are removed from B and collected in $B_{\neq\emptyset}$ together with their value $v^x = H\chi_S^C \cdot \xi(x)$, and then we group $B_{\neq\emptyset}$ by these values to obtain the new blocks $B_1, \ldots, B_\ell$ that we add to X/P.

Now we are ready to combine Split from Figure 8 with what we saw in Sections 4 and 5 and instantiate Algorithm 4.9 with the select routine from Example 4.8.(1), i.e. we have

$$q_{i+1} = \mathsf{select}(\kappa_{P_i}, \kappa_{Q_i}) \cdot \kappa_{P_i} = \chi_{S_i}^{C_i}, \tag{7.1}$$

where $2 \cdot |S_i| \leq |C_i|$, $S_i, C_i \subseteq X$ in line (1), and we replace line (3) of the algorithm by

$$X/P_{i+1} = \textsc{Split}(X/P_i, S_i), \tag{5.1'}$$

where the indices are merely intended to facilitate the analysis.

This yields the following more concretion of Algorithm 4.9:

Algorithm 7.7. Given the *encoding* of an H-coalgebra $\xi\colon X \to HX$ as input, do the following:

- Run Initialization (Figure 7).
- Iterate the following steps while X/P is properly finer than X/Q:
  (1) Pick a subblock S in X/P, that has at most half of the size of its compound block $C \in X/Q$, i.e. $S \subseteq C$ and $2 \cdot |S| \leq |C|$.
  (2) Split C into S and $C \setminus S$ in X/Q.
  (3) Call $\textsc{Split}(X/P, S)$ (Figure 8).

7.2. **Correctness.**

Lemma 7.8. *Assume that the invariants hold. Then after part (a) of Figure 8, for the given $S \subseteq C \in X/Q$ we have:*

(1) *For all $x \in X$:* $\mathsf{toSub}(x) = \{e \in E \mid e = x \xrightarrow{a} y, y \in S\}$

(2) *For all $x \in X$:* $\mathsf{fil}_S \cdot \mathfrak{b} \cdot \xi(x) = \mathcal{B}_{\mathrm{f}}(\pi_2 \cdot \mathsf{graph})(\mathsf{toSub}(x))(a)$.

(3) $\mathsf{M}\colon X/P \rightharpoonup H3$ *is a partial map defined by*

$$M(B) = H\chi^C_{\varnothing} \cdot \xi(x), \qquad \text{if } x \in B \text{ and there exists an } e = x \xrightarrow{a} y \text{ with } y \in S,$$

and $M(B)$ is undefined otherwise.

(4) *For each $B \in X/P$, we have a partial map* $\mathsf{mark}_B\colon B \rightharpoonup \mathbb{N}$ *defined by*

$$\mathsf{mark}_B(x) = \mathsf{lastW}(e) \qquad \text{if there exists some } e = x \xrightarrow{a} y \text{ with } y \in S,$$

and mark_B is undefined otherwise.

(5) *If defined on x,* $\mathsf{deref} \cdot \mathsf{mark}_B(x) = w(C, \xi(x))$.

(6) *If mark_B is undefined on x, then* $\mathsf{fil}_S(\mathfrak{b} \cdot \xi(x)) = \varnothing$ *and* $H\chi^C_S \cdot \xi(x) = H\chi^C_{\varnothing} \cdot \xi(x)$.

Proof. (1) By lines 2 and 11,

$$\begin{aligned}\mathsf{toSub}(x) &= \{e \in \mathsf{pred}(y) \mid y \in S, e = x \xrightarrow{a} y\}\\ &= \{e \in E \mid y \in S, e = x \xrightarrow{a} y\}.\end{aligned}$$

(2) $\mathsf{fil}_S(\mathfrak{b} \cdot \xi(x))(a) = \sum_{y \in S} (\mathfrak{b} \cdot \xi(x))(a, y) = \sum_{y \in S} |\{e \in E \mid e = x \xrightarrow{a} y\}|$

$$= |\{e \in E \mid e = x \xrightarrow{a} y, y \in S\}| = \mathcal{B}_{\mathrm{f}}(\pi_2 \cdot \mathsf{graph})(\mathsf{toSub}(x)).$$

(3) First, M is a partial map since for every block B, a pair (B, v) is added to M at most once in line 8 because when any node x from B occurs in line 3 for the first time we have that mark_B, $\mathsf{toSub}(x)$ are both empty, and they are both nonempty after line 11. By construction M is defined precisely for those blocks B which have at least one element x with an edge $e = x \xrightarrow{a} y$ to S. Let $C = [y]_{\kappa_Q} \in X/Q$. Then using Invariant (3) in the second step we see

that

$$\begin{aligned} M(B) &= \pi_2 \cdot \mathsf{update}(\varnothing, \mathsf{deref} \cdot \mathsf{lastW}(e)) \\ &= \pi_2 \cdot \mathsf{update}(\varnothing, w(C, \xi(x))) \\ &= \pi_2 \cdot \mathsf{update}(\mathsf{fil}_\varnothing(\flat \cdot \xi(x)), w(C, \xi(x))) \\ &\overset{(6.1)}{=} H\chi^C_\varnothing \cdot \xi(x) \end{aligned}$$

for the $e = x \xrightarrow{a} y$, $x \in B$, $y \in S$ that occurs first in the loop. Since $\mathsf{ker}(H\chi^C_\varnothing \cdot \xi) = \mathsf{ker}(H\chi_C \cdot \xi)$, Invariant (4) proves well-definedness.

(4) This is precisely how mark_B has been constructed; that it is a partial map follows since for every $x \in X$ a pair (x, n) is added to mark_B at most once in line 10 if $\mathsf{toSub}(x)$ is nonempty, and immediately after that $\mathsf{toSub}(x)$ becomes nonempty in line 11. Well-definedness follows from Invariant (2). Note that for every B on which M is undefined, the list mark_B is empty.

(5) If $\mathsf{mark}_B(x) = p_C$ is defined, then $p_C = \mathsf{lastW}(e)$ for some $e \in \mathsf{toSub}(x)$, and so $\mathsf{deref}(p_C) = \mathsf{deref} \cdot \mathsf{lastW}(e) = w(C, \xi(x))$ by Invariant (3).

(6) If $x \in B$ is not marked in B, then x was never mentioned in line 3. Hence, $\mathsf{toSub}(x) = \varnothing$, and we have

$$\mathsf{fil}_S(\flat \cdot \xi(x))(a) = |\{e \in E \mid e = x \xrightarrow{a} y, y \in S\}| = |\mathsf{toSub}(x)| = 0.$$

Furthermore we have

$$\begin{aligned} H\chi^C_S \cdot \xi(x) &= \pi_2 \cdot \mathsf{update}(\mathsf{fil}_S \cdot \flat \cdot \xi(x), w(C, \xi(x))) && \text{by (6.1)} \\ &= \pi_2 \cdot \mathsf{update}(\varnothing, w(C, \xi(x))) && \text{as just shown} \\ &= \pi_2 \cdot \mathsf{update}(\mathsf{fil}_\varnothing \cdot \flat \cdot \xi(x), w(C, \xi(x))) && \text{by definition} \\ &= H\chi^C_\varnothing \cdot \xi(x) && \text{by (6.1)} \end{aligned}$$

□

Theorem 7.9 (Correctness)**.** *If the invariants hold before invoking* Split*, then*

(i) Split *returns the correct partitions, that is,* $\textsc{Split}(X/P, X/Q, S \subseteq C \in X/Q)$ *refines* X/P *by* $H\chi^C_S \cdot \xi \colon X \to H3$*, i.e. all blocks in* X/P *are split by* $H\chi^C_S \cdot \xi$ *so that* P *is replaced by* $P \cap \mathsf{ker}(H\chi^C_S \cdot \xi)$*, and*

(ii) *upon termination of* Split *the invariants hold again.*

Proof. (i) We show that every block $B \in X/P$ is split by $H\chi^C_S \cdot \xi$, by case distinction whether $B \in \mathsf{M}$:

- If B is not in M, then mark_B is undefined everywhere, so for all $x \in B$, we have by Lemma 7.8(6) that $H\chi^C_S \cdot \xi(x) = H\chi^C_\varnothing \cdot \xi(x)$, which is the same for all $x \in B$ by invariant (4). Hence every B not in M stays unchanged when splitting by $H\chi^C_S \cdot \xi$.
- If $(B, v_\varnothing) \in \mathsf{M}$ (in line 12), then we show that

  $$B_{\neq\varnothing} = \{(x, H\chi^C_S \cdot \xi(x)) \mid x \in B, H\chi^C_S \cdot \xi(x) \neq v_\varnothing\}.$$

  This is sufficient for correctness, since $B_{\neq\varnothing}$ is split w.r.t. the second component (line 25) creating new blocks, and since all elements $x \in B$ not mentioned in $B_{\neq\varnothing}$ stay in B.

For this characterization of $B_{\neq\emptyset}$, suppose first that $x \in B$ is marked, i.e. we have $p_C = \mathsf{mark}_B(x)$ and lines 14–24 are executed. Then we have

$$\begin{aligned}(w_S^x, v^x, w_{C\setminus S}^x) &= \mathsf{update}(\mathcal{B}_{\mathrm{f}}(\pi_2 \cdot \mathsf{graph})(\mathsf{toSub}[x]), \mathsf{deref}[p_C]) && \text{by line 16}\\ &= \mathsf{update}(\mathsf{fil}_S \cdot \flat \cdot \xi(x), w(C, \xi(x))) && \text{by Lemma 7.8 (2), (5)}\\ &= (w(S,\xi(x)), H\chi_S^C \cdot \xi(x), w(C \setminus S, \xi(x))) && \text{by (6.1).}\end{aligned}$$

Thus, if $H\chi_S^C \cdot \xi(x) = v_\emptyset$, (x, v^x) is not added to $B_{\neq\emptyset}$ (line 21), and otherwise x is correctly removed from B and added to $B_{\neq\emptyset}$ in line 24.

Now suppose that $x \in B$ is not marked, then x is not added to $B_{\neq\emptyset}$ and so we have to show that $v_\emptyset = H\chi_S^C \cdot \xi(x)$. We have $v_\emptyset = H\chi_\emptyset^C \cdot \xi(x')$ for some $x' \in B$ by Lemma 7.8(3) and $H\chi_S^C \cdot \xi(x) = H\chi_\emptyset^C \cdot \xi(x)$ by Lemma 7.8(6). Since $\mathsf{ker}(H\chi_\emptyset^C \cdot \xi) = \mathsf{ker}(H\chi_C \cdot \xi)$ and $(x, x') \in \mathsf{ker}(H\chi_C \cdot \xi)$ by invariant (4), we have $H\chi_\emptyset^C \cdot \xi(x') = H\chi_\emptyset^C \cdot \xi(x)$. Thus,

$$v_\emptyset = H\chi_\emptyset^C \cdot \xi(x') = H\chi_\emptyset^C \cdot \xi(x) = H\chi_S^C \cdot \xi(x).$$

(ii) We denote the former values of $P, Q, \mathsf{deref}, \mathsf{lastW}$ using the subscript old.

(1) It is easy to see that $\mathsf{toSub}(x)$ becomes nonempty in line 11 only for marked x, and for those x it is emptied again in line 20.

(2) Take $e_1 = x_1 \xrightarrow{a_1} y_1$, $e_2 = x_2 \xrightarrow{a_2} y_2$.

$\Rightarrow$ Assume $\mathsf{lastW}(e_1) = \mathsf{lastW}(e_2)$. If $\mathsf{lastW}(e_1) = p_S$ is assigned in line 19 for some marked x, then $\mathsf{lastW}(e_2)$ must be assigned to p_S in the same **for** loop since p_S is the address of a new memory cell (line 18). Hence, $e_1, e_2 \in \mathsf{toSub}(x)$, which implies that $x_1 = x = x_2$ and $y_1, y_2 \in S \in X/Q$. Otherwise, $\mathsf{lastW}(e_1) = \mathsf{lastW}_{\mathsf{old}}(e_1)$ and so $\mathsf{lastW}_{\mathsf{old}}(e_1) = \mathsf{lastW}_{\mathsf{old}}(e_2)$ and the desired property follows from the invariant for $\mathsf{lastW}_{\mathsf{old}}$.

$\Leftarrow$ If $x_1 = x_2$ and $y_1, y_2 \in D \in X/Q$, where recall that $X/Q = X/Q_{\mathsf{old}} \setminus \{C\} \cup \{S, C \setminus S\}$, then we perform a case distinction on D. If $D = S$, then $e_1, e_2 \in \mathsf{toSub}(x)$ (Lemma 7.8(1)) and so both entries in the array are set to the same value $\mathsf{lastW}(e_1) = \mathsf{lastW}(e_2) = p_S$ in line 19. If $D \neq S$, then $e_1, e_2 \notin \mathsf{toSub}(x)$ and so $\mathsf{lastW}[e_1]$ and $\mathsf{lastW}[e_2]$ stay unchanged. Hence, we have

$$\mathsf{lastW}(e_1) = \mathsf{lastW}_{\mathsf{old}}(e_1) = \mathsf{lastW}_{\mathsf{old}}(e_2) = \mathsf{lastW}(e_2).$$

(3) Let $e = x \xrightarrow{a} y$, $D := [y]_{\kappa_Q} \in X/Q$. We again perform a case distinction on D:

$$\begin{array}{lcl} D = S & \Rightarrow & \mathsf{deref} \cdot \mathsf{lastW}(e) = \mathsf{deref}(p_S) = w_S^x = w(S, \xi(x)),\\ D = C \setminus S & \Rightarrow & \mathsf{deref} \cdot \mathsf{lastW}(e) = \mathsf{deref}(p_C) = w_{C\setminus S}^x = w(C \setminus S, \xi(x)),\\ D \in X/Q_{\mathsf{old}} \setminus \{C\} & \Rightarrow & \mathsf{deref} \cdot \mathsf{lastW}(e) = \mathsf{deref}_{\mathsf{old}} \cdot \mathsf{lastW}(e)_{\mathsf{old}} = w(D, \xi(x)).\end{array}$$

Note that the first equation in the second case holds due to lines 10 and 14. For the first two cases note that $w_S^x = w(S, \xi(x))$ and $w_{C\setminus S}^x = w(C \setminus S, \xi(x))$ by line 16, Lemma 7.8, items (2) and (5), and by the axiom for update in (6.1).

(4) Take $x_1, x_2 \in B' \in X/P$ and $D \in X/Q$ and let $B := [x_1]_{P_{\mathsf{old}}} = [x_2]_{P_{\mathsf{old}}} \in X/P_{\mathsf{old}}$. By case distinction on $\mathsf{M}(B)$ we first show that $H\chi_S^C \cdot \xi(x_1) = H\chi_S^C \cdot \xi(x_2)$.

(a) If $\mathsf{M}(B)$ is defined, then $H\chi_S^C \cdot \xi(x_1) = H\chi_S^C \cdot \xi(x_2)$ – for otherwise x_1 and x_2 would have been put into different blocks in line 25.

(b) If $\mathsf{M}(B)$ is undefined, then mark_B is undefined everywhere, in particular for x_1 and x_2. Then, by Lemma 7.8(6) we have $H\chi_S^C \cdot \xi(x_i) = H\chi_\varnothing^C \cdot \xi(x_i)$ for $i = 1, 2$. Since $C \in X/Q_{\mathsf{old}}$ and $\mathsf{ker}(H\chi_\varnothing^C \cdot \xi) = \mathsf{ker}(H\chi_C \cdot \xi)$, we have $H\chi_\varnothing^C \cdot \xi(x_1) = H\chi_\varnothing^C \cdot \xi(x_2)$ by invariant (4), and so $(x_1, x_2) \in \mathsf{ker}(H\chi_S^C \cdot \xi)$.

We can conclude the invariant by another case distinction on D:

$$\begin{array}{lcl} D = S & \Rightarrow & (x_1, x_2) \in \mathsf{ker}(H\chi_S^C \cdot \xi) \subseteq \mathsf{ker}(\overbrace{H(=2) \cdot H\chi_S^C}^{H\chi_S = H\chi_D} \cdot \xi), \\ D = C \setminus S & \Rightarrow & (x_1, x_2) \in \mathsf{ker}(H\chi_S^C \cdot \xi) \subseteq \mathsf{ker}(\underbrace{H(=1) \cdot H\chi_S^C}_{H\chi_{C\setminus S} = H\chi_D} \cdot \xi), \\ D \in X/Q_{\mathsf{old}} \setminus \{C\} & \Rightarrow & (x_1, x_2) \in \mathsf{ker}(H\chi_D \cdot \xi), \end{array}$$

where we use Remark 2.4(1) and where the last statement holds by invariant (4) for X/Q_{old}. □

Corollary 7.10. *Algorithm 7.7 computes the simple quotient of a given finite coalgebra $\xi\colon X \to HX$.*

Proof. Indeed, by Theorem 7.9(i) and (7.1) we see that (5.1') is equivalent to (5.1):

$$P_{i+1} = P_i \cap \mathsf{ker}(H\chi_{S_i}^{C_i} \cdot \xi) = P_i \cap \mathsf{ker}(Hq_{i+1} \cdot \xi).$$

By Corollary 5.21, this is equivalent to the original line (3) in Algorithm 4.9, since the employed "select the smaller half" routine respects compound blocks and H is zippable. The desired result thus follows from Theorem 4.20. □

7.3. **Efficiency.** Having established correctness of Split, we next analyse its time complexity. We first analyse lines $1 - 24$, then the complexity of the grouping operation in line 25, and finally the overall complexity of the algorithm, accumulating the time for all Split invocations.

Lemma 7.11. *Lines $1 - 24$ in* Split *run in time $\mathcal{O}(\sum_{y \in S} |\mathsf{pred}(y)|)$.*

Proof. The loop in line 2 has $\sum_{y \in S} |\mathsf{pred}(y)|$ iterations, each consisting of constantly many operations taking constant time. Since each loop appends one element to some initially empty $\mathsf{toSub}(x)$, we have

$$\sum_{y \in S} |\mathsf{pred}(y)| = \sum_{x \in X} |\mathsf{toSub}(x)|.$$

In the body of the loop starting in line 14, the only statements not running in constant time are $\ell := \mathcal{B}_{\mathrm{f}}(\pi_2 \cdot \mathsf{graph})(\mathsf{toSub}(x))$ (line 15), $\mathsf{update}(\ell, \mathsf{deref}(p_C))$ (line 16), and the loop in line 19; each of these require time linear in the length of $\mathsf{toSub}(x)$. The loop in line 14 has at most one iteration per $x \in X$. Hence, since each x is contained in at most one block B from line 12, the overall complexity of line 12 to 24 is at most $\sum_{x \in X} |\mathsf{toSub}(x)| = \sum_{y \in S} |\mathsf{pred}(y)|$, as desired. □

In the grouping operation in line 25, it is not enough to group the elements using a sorting algorithm. Instead we need to preprocess the elements and extract a possible majority candidate.

Definition 7.12. When grouping Z by $f\colon Z \to Z'$ we call an element $p \in Z'$ a *possible majority candidate* (PMC) if either

$$|\{z \in Z \mid f(z) = p\}| \geq |\{z \in Z \mid f(z) \neq p\}| \tag{7.2}$$

or if no element in Z' fulfilling (7.2) exists.

A PMC can be computed in linear time [Bac86, Sect. 4.3.3]. When grouping Z by f using a PMC, one first determines a PMC $p \in Z'$, and then one only sorts and groups $\{z \mid f(z) \neq p\}$ by f using an $\mathcal{O}(n \cdot \log n)$ sorting algorithm.

Remark 7.13. In the following lemmas we again index data by the iterations $1 \leq i \leq k$ in Algorithm 7.7. That means we consider $S_i \subseteq C_i \in X/Q_{i+1}$ such that

$$X/Q_{i+1} = X/Q_i \setminus \{C_i\} \cup \{S_i, C_i \setminus S_i\}. \tag{7.3}$$

Lemma 7.14. *Summing over all* Split *invocations in Algorithm 7.7, the total time spent on grouping $B_{\neq\varnothing}$ using a PMC is in $\mathcal{O}(|E| \cdot \log |X|)$.*

The proof is a generalization of that for the weighted setting of Valmari and Franceschinis [VF10, Lemma 5]:

Proof. We shall prove that for $S_i \subseteq C_i \in X/Q_i$, $0 \leq i < k$, in Remark 7.13, the overall time spent on grouping the $B_{\neq\varnothing}$ in all the runs of Split is in $\mathcal{O}(|E| \cdot \log |X|)$. The partition returned by the algorithm is $X/P_k = X/Q_k$ and is obtained after Split has been called k-many times.

In the first part of the proof, we define subsets $\mathcal{L}_B, \mathcal{M}_B \subseteq B_{\neq\varnothing}$ which are generalized version of the left-hand and middle subblocks of a block B in Figure 6 for a $\mathcal{P}_\mathrm{f}$-coalgebra, i.e. those state with successors in S but not in $C \setminus S$ and those with successors in both. We then give an equivalent characterization of $\mathcal{L}_B$ and $\mathcal{M}_B$.

In the second part, we use this characterization and a PMC to argue that sorting each $B_{\neq\varnothing}$ is bounded by $2 \cdot |\mathcal{M}_B| \cdot \log(2 \cdot |\mathcal{M}_B|)$. Since we assume that comparing two elements of $H3$ runs in constant time, the time needed for sorting amounts to the number of comparisons needed while sorting, i.e. $\mathcal{O}(n \cdot \log n)$ many.

Finally, in the third part, we use this to obtain the desired overall complexity.

(1) For a $(B, v_\varnothing) \in \mathsf{M}$ in the iteration i consider $B_{\neq\varnothing}$. We define

- the *left block* $\mathcal{L}^i_B := \{x \in B \mid H\chi^{C_i}_{C_i} \cdot \xi(x) = H\chi^{C_i}_{S_i} \cdot \xi(x) \neq H\chi^{C_i}_{\varnothing} \cdot \xi(x)\}$, and
- the *middle block* $\mathcal{M}^i_B := \{x \in B \mid H\chi^{C_i}_{C_i} \cdot \xi(x) \neq H\chi^{C_i}_{S_i} \cdot \xi(x) \neq H\chi^{C_i}_{\varnothing} \cdot \xi(x)\}$.

Now, first note that

$$(x, v^x) \in B_{\neq\varnothing} \text{ iff } v^x = H\chi^{C_i}_{S_i} \cdot \xi(x) \neq v_\varnothing, \qquad \text{and} \qquad v_\varnothing = H\chi^{C}_{\varnothing} \cdot \xi(x), \tag{7.4}$$

where the latter holds by Lemma 7.8(3). This implies that the set of first components of the pairs in $B_{\neq\varnothing}$ is $\mathcal{L}^i_B \cup \mathcal{M}^i_B$. If $x \in B$ has no edge to S_i, then it is not marked, and so $H\chi^{C_i}_{S_i} \cdot \xi(x) = H\chi^{C_i}_{\varnothing} \cdot \xi(x)$, by Lemma 7.8(6); by contraposition, every $x \in \mathcal{L}^i_B \cup \mathcal{M}^i_B$ has some edge into S_i. We can make a similar observation for $C_i \setminus S_i$. If $x \in B$ has no edge to $C_i \setminus S_i$, then $\mathsf{fil}_{S_i}(\flat \cdot \xi(x)) = \mathsf{fil}_{C_i}(\flat \cdot \xi(x))$ by the definition of fil_{S_i}, and therefore we have:

$$\begin{aligned} H\chi^{C_i}_{C_i} \cdot \xi(x) &\overset{(6.1)}{=} \pi_2 \cdot \mathsf{update}(\mathsf{fil}_{C_i}(\flat \cdot \xi(x)), w(C_i, \xi(x))) \\ &= \pi_2 \cdot \mathsf{update}(\mathsf{fil}_{S_i}(\flat \cdot \xi(x)), w(C_i, \xi(x))) \overset{(6.1)}{=} H\chi^{C_i}_{S_i} \cdot \xi(x). \end{aligned}$$

By contraposition, all $x \in \mathcal{M}^i_B$ have an edge to $C_i \setminus S_i$.

Note that $\ker(H\chi^{C_i}_{C_i} \cdot \xi) = \ker(H\chi_{C_i} \cdot \xi)$; indeed, to see this use Remark 2.4(2), that H preserves monomorphisms, and that $\chi^{C_i}_{C_i} = m \cdot \chi_{C_i}$, where $m\colon 2 \cong \{0, 2\} \hookrightarrow \{0, 1, 2\} = 3$ is

the inclusion map. Since $\mathcal{L}_B^i \subseteq B \in X/P_i$ and $C_i \in X/Q_i$, we conclude from invariant (4) that there is an $\ell_B^i \in H3$ such that $\ell_B^i = H\chi_{C_i}^{C_i} \cdot \xi(x)$ for all $x \in B$. Using (7.4) we therefore obtain

$$\mathcal{L}_B^i = \{x \mid (x, v^x) \in B_{\neq\emptyset}, v^x = \ell_B^i\} \quad \text{and} \quad \mathcal{M}_B^i = \{x \mid (x, v^x) \in B_{\neq\emptyset}, v^x \neq \ell_B^i\}.$$

We have also seen that every $x \in \mathcal{L}_B^i$ has an edge to S_i and every $x \in \mathcal{M}_B^i$ has both an edge to S_i and $C_i \setminus S_i$.

(2) We prove that sorting $B_{\neq\emptyset}$ in the iteration i is bound by $2 \cdot |\mathcal{M}_B^i| \cdot \log(2 \cdot |\mathcal{M}_B^i|)$ by case distinction on the possible majority candidate:

- If ℓ_B^i is the possible majority candidate, then the sorting of $B_{\neq\emptyset}$ sorts precisely $\mathcal{M}_B^i$ which indeed amounts to
$$|\mathcal{M}_B^i| \cdot \log(|\mathcal{M}_B^i|) \leq 2 \cdot |\mathcal{M}_B^i| \cdot \log(2 \cdot |\mathcal{M}_B^i|).$$
- If ℓ_B^i is not the possible majority candidate, then $|\mathcal{L}_B^i| \leq |\mathcal{M}_B^i|$. In this case sorting $B_{\neq\emptyset}$ is bounded by
$$(|\mathcal{L}_B^i| + |\mathcal{M}_B^i|) \cdot \log(|\mathcal{L}_B^i| + |\mathcal{M}_B^i|) \leq 2 \cdot |\mathcal{M}_B^i| \cdot \log(2 \cdot |\mathcal{M}_B^i|).$$

(3) Finally, we show that we obtain the desired overall time complexity. Let the number of blocks to which x has an edge be denoted by

$$\sharp_Q^i(x) = |\{D \in X/Q_i \mid e = x \xrightarrow{a} y, y \in D\}|, \quad i \geq 0.$$

Clearly, this number is bounded by the number of outgoing edges of x, i.e. $\sharp_Q^i(x) \leq |\flat \cdot \xi(x)|$, and so

$$\sum_{x \in X} \sharp_Q^i(x) \leq \sum_{x \in X} |\flat \cdot \xi(x)| = |E|.$$

Define

$$\sharp_{\mathcal{M}}^i(x) = |\{0 \leq j < i \mid x \text{ is in some } \mathcal{M}_B^j\}|, \quad i \geq 0.$$

If in the iteration $i \geq 0$, x is in a middle block $\mathcal{M}_B^i$, then $\sharp_{\mathcal{M}}^{i+1}(x) = \sharp_{\mathcal{M}}^i(x) + 1$ and $\sharp_Q^{i+1}(x) = \sharp_Q^i(x) + 1$, by (7.3) and since x has both an edge to S_i and $C_i \setminus S_i$. Otherwise, if x is not in any middle block in iteration i, then $\sharp_{\mathcal{M}}^{i+1}(x) = \sharp_{\mathcal{M}}^i(x)$ and $\sharp_Q^{i+1}(x) \geq \sharp_Q^i(x)$. This implies that for all $i \geq 0$, $\sharp_{\mathcal{M}}^i(x) \leq \sharp_Q^i(x)$, and therefore

$$\sum_{x \in X} \sharp_{\mathcal{M}}^i(x) \leq \sum_{x \in X} \sharp_Q^i(x) \leq |E|.$$

Let T denote the total number of middle blocks $\mathcal{M}_B^i$, $0 \leq i < k$, such that B is contained in M in iteration i, and let $\mathcal{M}_t$, $1 \leq t \leq T$, be the t^{th} middle block. The sum of the sizes of all middle blocks is the same as the number of times each $x \in X$ was contained in a middle block, i.e.

$$\sum_{t=1}^{T} |\mathcal{M}_t| = \sum_{x \in X} \sharp_{\mathcal{M}}^k(x) \leq |E|.$$

Using the previous bounds and $|\mathcal{M}_t| \leq |X|$, we now obtain

$$\sum_{t=1}^{T} 2 \cdot |\mathcal{M}_t| \cdot \log(2 \cdot |\mathcal{M}_t|) \leq \sum_{t=1}^{T} 2 \cdot |\mathcal{M}_t| \cdot \log(2 \cdot |X|) = 2 \cdot \big(\sum_{t=1}^{T} |\mathcal{M}_t| \big) \cdot \log(2 \cdot |X|)$$
$$\leq 2 \cdot |E| \cdot \log(2 \cdot |X|) = 2 \cdot |E| \cdot \log(|X|) + 2 \cdot |E| \cdot \log(2) \in \mathcal{O}\big(|E| \cdot \log(|X|)\big). \qquad \square$$

Lemma 7.15. (1) *For each $y \in X$, $|\{i < k \mid y \in S_i\}| \leq \log_2 |X| + 1$.*
(2) *The total run-time of all invocations of* $\textsc{Split}(X/P_i, S_i)$, $0 \leq i < k$, *is in $\mathcal{O}(|E| \cdot \log |X|)$.*

Proof. (1) We know from Lemma 4.11 that Q_j is finer than Q_i for every $j > i$. Moreover, by (7.3), we have $S_i \in X/Q_{i+1}$. For every $i < j$ with $y \in S_i$ and $y \in S_j$, we know that $C_j \subseteq S_i$ since C_j is the block containing y in the refinement X/Q_j of X/Q_{i+1} in which S_i contains y. Hence, we have $2 \cdot |S_j| \leq |C_j| \leq |S_i|$. Now let $i_1 < \ldots < i_n$ be all the elements in $\{i < k \mid y \in S_i\}$. Since $y \in S_{i_1}, \ldots, y \in S_{i_n}$, we have $2^{n-1} \cdot |S_{i_n}| \leq |S_{i_1}|$. Thus

$$|\{i < k \mid y \in S_i\}| = n = \log_2(2^{n-1}) + 1$$
$$\leq \log_2(2^{n-1} \cdot |S_{i_n}|) + 1 \leq \log_2 |S_{i_1}| + 1 \leq \log_2 |X| + 1,$$

where the last inequality holds since $S_{i_1} \subseteq X$.

(2) By Lemma 7.11, the first term below gives the total run-time in the $\mathcal{O}$-calculus, and we continue to reason in that calculus (note that the inner sums on the right-hand side of the first line are indexed by S_i, whence the $\ni$-symbol):

$$\sum_{0 \leq i < k} \sum_{y \in S_i} |\mathsf{pred}(y)| = \sum_{y \in X} \sum_{\substack{0 \leq i < k \\ S_i \ni y}} |\mathsf{pred}(y)| = \sum_{y \in X} \big(|\mathsf{pred}(y)| \cdot \sum_{\substack{0 \leq i < k \\ S_i \ni y}} 1 \big)$$
$$= \sum_{y \in X} \big(|\mathsf{pred}(y)| \cdot |\{i < k \mid y \in S_i\}| \big)$$
$$\leq \sum_{y \in X} \big(|\mathsf{pred}(y)| \cdot (\log |X| + 1) \big)$$
$$= \big(\sum_{y \in X} |\mathsf{pred}(y)| \big) \cdot (\log |X| + 1)$$
$$= |E| \cdot (\log |X| + 1),$$

where the inequality holds by the first part of our lemma. $\square$

By Lemmas 7.6 and 7.15(2), we obtain our main result:

Theorem 7.16. *Given a zippable functor $H \colon \mathsf{Set} \to \mathsf{Set}$ with a refinement interface, whose* update *and* init *functions can be computed in linear time, Algorithm 7.7 computes the quotient modulo behavioural equivalence of a given coalgebra $\xi \colon X \to HX$ with $n = |X|$ states and $m = \sum_{x \in X} |\flat \cdot \xi(x)|$ edges in time $\mathcal{O}((m + n) \cdot \log n)$.*

If the coalgebra is not too sparse, i.e. every state has at least one in- or outgoing edge, $2 \cdot m \geq n$, then the complexity is $\mathcal{O}(m \cdot \log n)$, the bound typically seen in the literature for efficient algorithms for bisimilarity minimization of transition systems $\xi \colon X \to \mathcal{P}_{\mathrm{f}} X$ or weighted systems $\xi \colon X \to \mathbb{R}^{(X)}$. Unlike those algorithms we do not directly admit an initial partition as a parameter; but switching from a functor G to $X/\mathcal{I} \times G$ (cf. Remark 4.15) we

can equip the generic algorithm with this additional parameter while maintaining the same $\mathcal{O}(m \cdot \log n)$ complexity:

Remark 7.17. There are two ways to handle functors of type $H = X/\mathcal{I} \times G$. First, we can modify the functor interface as follows:

$$\begin{array}{lcl} G1 & \mapsto & X/\mathcal{I} \times G1 \\ W & \mapsto & X/\mathcal{I} \times W \end{array} \qquad \begin{array}{lcl} \mathsf{b} & \mapsto & X/\mathcal{I} \times GX \xrightarrow{\pi_2} GX \xrightarrow{\mathsf{b}} \mathcal{B}_{\mathrm{f}}(A \times X) \\ \mathsf{init} & \mapsto & \mathsf{id}_X/\mathcal{I} \times \mathsf{init} \end{array}$$

update is replaced by the following

$$\begin{array}{ccc} (\mathcal{B}_{\mathrm{f}}(A) \times W) \times X/\mathcal{I} & \xrightarrow{\mathsf{update} \times X/\mathcal{I}} & (W \times G3 \times W) \times X/\mathcal{I} \\ \cong \uparrow & & \downarrow \\ \mathcal{B}_{\mathrm{f}}(A) \times (X/\mathcal{I} \times W) & & (X/\mathcal{I} \times W) \times (X/\mathcal{I} \times G3) \times (X/\mathcal{I} \times W) \end{array}$$

where the first and the last morphism are the obvious ones. A second approach is to decompose the functor into $X/\mathcal{I} \times (-)$ and G, moving to the multisorted setting, see Section 8 for more details, in particular Example 8.19. Both methods have no effect on the complexity.

Example 7.18. As instances of our algorithm, we obtain the following standard examples for partition refinement algorithms:

(1) For $H = X/\mathcal{I} \times \mathcal{P}_{\mathrm{f}}$, we obtain the classical Paige-Tarjan algorithm [PT87] (with initial partition $X/\mathcal{I}$), with the same complexity $\mathcal{O}((m+n) \cdot \log n)$.

Recall from Example 2.11 that the equivalence computed by the Paige-Tarjan algorithm – bisimilarity – is precisely behavioural equivalence for the powerset functor $\mathcal{P}_{\mathrm{f}}$, also when an initial partition is taken into account (Remark 4.15). $\mathcal{P}_{\mathrm{f}}$ and $X/\mathcal{I} \times \mathcal{P}_{\mathrm{f}}$ are zippable functors (Example 5.7, Lemma 5.5), and we have refinement interfaces for them (Example 6.6, Remark 7.17) that realize the desired time complexity bound (Example 6.11). So the instantiation of Algorithm 7.7 computes the bisimilarity relation on an input coalgebra with n states and m edges in time $\mathcal{O}((m+n) \cdot \log n)$ by Theorem 7.16.

The instantiation of Algorithm 7.7 for the refinement interface of $\mathcal{P}_{\mathrm{f}}$ is nearly identical to the Paige-Tarjan algorithm [PT87], informally described in Example 6.3. For example, both maintain references from the edges to the integer counters. However, the (three way-)split of a block is simpler in the specific implementation than in Split (Line 25), because $|\mathcal{P}_{\mathrm{f}}3| = 8$ is finite and in fact in Line 25 at most three different values can occur (namely the three cases from Example 6.3).

For an example run, see Example 3.3 and Figure 1a.

(2) For $HX = X/\mathcal{I} \times \mathbb{R}^{(X)}$, we solve Markov chain lumping with an initial partition $X/\mathcal{I}$ in time $\mathcal{O}((m+n) \cdot \log n)$, like the best known algorithm by Valmari and Franceschinis [VF10]:

Coalgebraic behavioural equivalence for $\mathbb{R}^{(-)}$ captures precisely weighted bisimilarity (Example 2.11), and $\mathbb{R}^{(-)}$ is zippable (Example 5.7) and has a refinement interface (Example 6.7) with the required complexity bounds (Example 6.11). So the instantiation of Algorithm 7.7 computes weighted bisimilarity for an input coalgebra with n states and m edges in $\mathcal{O}((m+n) \cdot \log n)$ by Theorem 7.16.

The algorithm by Valmari and Franceschinis [VF10] is essentially that of Split (Figure 8), after making simplifications using properties of $\mathbb{R}^{(-)}$. For instance, Valmari and Franceschinis

do not need to keep the accumulated weights from states to blocks in memory since the weights can be computed from the labels within the first loop of Split.

For an example run, see Example 3.3 and Figure 1b.

(3) The functor $H = \mathcal{B}_{\mathrm{f}}$ has a refinement interface (Example 6.7) with the desired run-time (Example 6.11), and thus Algorithm 7.7 computes behavioural equivalence on a $\mathcal{B}_{\mathrm{f}}$-coalgebra with n edges and m states in time $\mathcal{O}((m+n)\cdot \log n)$.

Partition refinement for "undirected" $\mathcal{B}_{\mathrm{f}}$-coalgebras is known as *colour refinement* and called the *1-dimensional Weisfeiler-Lehman Algorithm* (WL), where undirected means that $x \xrightarrow{n} y$ iff $y \xrightarrow{n} x$. Colour refinement is an important subroutine in graph isomorphism checking (and was originally conjectured to check graph isomorphism, see e.g. [CFI92, Wei76, BBG17]). Its input is an undirected graph (V, E), i.e. E is a set of two-element subsets of V. Then color refinement is just partition refinement on the $\mathcal{B}$-coalgebra $\xi\colon V \to \mathcal{B}V$ defined by $\xi(u)(v) = 1$ if $\{u, v\} \in E$ and $\xi(u)(v) = 0$ otherwise (i.e. one introduces two directed edges per undirected edge as usual). Our algorithm runs in $\mathcal{O}((m+n)\cdot \log n)$, matching the run-time of the optimal algorithm by Berkholz, Bonsma, and Grohe [BBG17], and improving the run-time of $\mathcal{O}(m \cdot n)$ of a previous algorithm [SSvL$^+$11].

(4) Hopcroft's classical automata minimization [Hop71] is obtained by $HX = 2 \times X^A$, with running time $\mathcal{O}(n \cdot \log n)$ for the binary input alphabet $A = \{0, 1\}$ (Gries [Gri73] and Knuutila [Knu01] present this algorithm for arbitrary finite input alphabets A which are not fixed but part of the input of minimization).

For a fixed finite alphabet A, the functor $HX = 2 \times X^A$ – and also any other polynomial functor with bounded arity – is zippable (Lemma 5.5) and has a refinement interface (Example 6.7) fulfilling the complexity bound (Example 6.11). So Algorithm 7.7 computes behavioural equivalence for an input coalgebra with n states and m edges in time $\mathcal{O}((m+n)\cdot \log n)$ by Theorem 7.16. The functor encoding for polynomial functors (Example 6.2) encodes one occurrence of a k-ary operation symbol by k edges. Since we assume the signature to have bounded arity, the maximal k is independent of the coalgebra size and thus a constant factor, and so $m \leq \max(k) \cdot n \in \mathcal{O}(n)$. So the run-time of Algorithm 7.7 simplifies to $\mathcal{O}(n \cdot \log n)$.

Note that the mentioned automata minimization algorithms [Hop71] are different from Algorithm 7.7 because they perform a refinement step for each input symbol $a \in A$, whereas Split refines a block w.r.t. all labels, since it considers all edges into the subblock $S \subseteq C$ of interest.

If A is variable and part of the input, we can fit the minimization in the present generic framework via the following Section 8 on composite functors and multisorted sets (see Example 8.19 below).

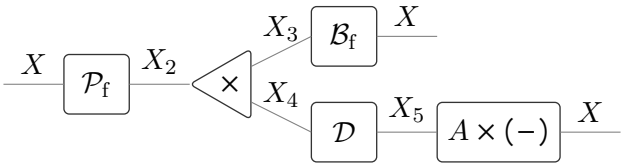
8. Modularity via Multisorted Coalgebra

We next describe how to minimize systems that mix different transition types. For example, recall from Example 2.6(5) that Segala systems mix non-deterministic and probabilistic branching in a way that makes them coalgebras for the composite functor $X \mapsto \mathcal{P}_{\mathrm{f}}(\mathcal{D}(A \times X))$ (or $X \mapsto \mathcal{P}_{\mathrm{f}}(A \times \mathcal{D}X)$ in the case of simple Segala systems, respectively). For our purposes, such functors raise the problem that zippable functors are not closed under composition. In the following, we show how to deal with this issue by moving from composite functors to multisorted coalgebras in the spirit of previous work on modularity in coalgebraic

logic [SP11]. Subsequently, the arising multisorted coalgebras are transformed back to singlesorted coalgebras by coproduct formation.

8.1. **Explicit intermediate states via multisortedness.** The transformation from coalgebras for composite functors into multisorted coalgebras is best understood by example:

Example 8.1. The functor $TX = \mathcal{P}_{\mathrm{f}}(\mathcal{B}_{\mathrm{f}}X \times \mathcal{D}(A \times X))$ can be visualized as



where we label the inner connections with fresh names X_2, X_3, X_4, X_5. From this visualization, we derive a functor $\bar{T}\colon \mathsf{Set}^5 \to \mathsf{Set}^5$:

$$\begin{aligned}\bar{T}\,(X,\quad X_2,\qquad X_3,\ X_4,\quad X_5\quad) = \\ (\mathcal{P}_{\mathrm{f}}X_2,\ X_3 \times X_4,\ \mathcal{B}_{\mathrm{f}}X,\ \mathcal{D}X_5,\ A \times X).\end{aligned}$$

Formal definitions following [SP11] are as follows.

Definition 8.2. Given a set $\mathcal{H}$ of mono-preserving and finitary functors $H\colon \mathsf{Set}^k \to \mathsf{Set}$ (with possibly different arities $k < \omega$), let $T\colon \mathsf{Set} \to \mathsf{Set}$ be a functor generated by the grammar

$$T ::= (-) \mid H(T, \dots, T)$$

where H ranges over $\mathcal{H}$ and $(-)$ is the argument, i.e. $T = (-)$ is the functor $TX = X$. By inspecting the structure of such a term T, we can define a functor $\bar{T}\colon \mathsf{Set}^n \to \mathsf{Set}^n$, where n is the number of non-leaf subterms of T (i.e. subterms of T including T itself but not $(-)$). Let f be a bijection from non-leaf subterms of T to natural numbers $\{1, \dots, n\}$, with $f(T) = 1$, and write $f(-) = 1$ to simplify notation (still, $f^{-1}(1) = T$). The *flattening* of T is $\bar{T}\colon \mathsf{Set}^n \to \mathsf{Set}^n$, given by

$$(\bar{T}(X_1, \dots, X_n))_i = H(X_{f(G_1)}, \dots, X_{f(G_k)}) \quad \text{where } f^{-1}(i) = H(G_1, \dots, G_k).$$

Intuitively speaking, we introduce a sort for each wire in the visualization but identify the outermost wires (labelled X in Example 8.1). Note that we keep track of duplicates, so e.g. $TX = \mathcal{P}_{\mathrm{f}}X \times \mathcal{P}_{\mathrm{f}}X$ has three subterms: $(-) \times (-)$, the left hand and the right hand $\mathcal{P}_{\mathrm{f}}$, so $n = 3$ and $\bar{T}(X_1, X_2, X_3) = (X_2 \times X_3, \mathcal{P}_{\mathrm{f}}X_1, \mathcal{P}_{\mathrm{f}}X_1)$.

In the remainder of this section we write T for functors defined according to the grammar in Definition 8.2 but continue to write H for functors in general (in particular for elements of $\mathcal{H}$).

Example 8.3. In Example 8.1 the functor T is built from the set $\mathcal{H}$ of functors containing

$$\mathcal{P}_{\mathrm{f}}, \mathcal{B}_{\mathrm{f}}, \mathcal{D}, A \times (-)\colon \mathsf{Set} \to \mathsf{Set} \qquad \times\colon \mathsf{Set}^2 \to \mathsf{Set},$$

and the term $T = \mathcal{P}_{\mathrm{f}}(\mathcal{B}_{\mathrm{f}}(-) \times \mathcal{D}(A \times (-)))$ has the following non-leaf subterms, implicitly defining the bijection f:

1. $\mathcal{P}_{\mathrm{f}}(\mathcal{B}_{\mathrm{f}}(-) \times \mathcal{D}(A \times (-)))$ 2. $\mathcal{B}_{\mathrm{f}}(-) \times \mathcal{D}(A \times (-))$ 3. $\mathcal{B}_{\mathrm{f}}(-)$
4. $\mathcal{D}(A \times (-))$ 5. $A \times (-)$.

Then, $\bar{T}\colon \mathsf{Set}^5 \to \mathsf{Set}^5$ is defined by

$$\bar{T}(X_1, X_2, X_3, X_4, X_5) = (\mathcal{P}_\mathrm{f} X_2, X_3 \times X_4, \mathcal{B}_\mathrm{f} X_1, \mathcal{D} X_5, A \times X_1).$$

Now a coalgebra for the flattening $\bar{T}$ of a functor term T is a family of maps

$$\xi_i \colon X_i \to H(X_{f(G_1)}, \ldots, X_{f(G_k)}) \quad \text{where } f^{-1}(i) = H(G_1, \ldots, G_k).$$

For example, given a T-coalgebra $\xi\colon X \to TX$, the morphism $(\xi, \mathsf{id}, \ldots, \mathsf{id})$ in Set^n is a coalgebra for the flattening $\bar{T}\colon \mathsf{Set}^n \to \mathsf{Set}^n$ of T. Note that this defines the first sort to be X and implicitly defines the other sorts. This mapping defines a functor $\mathsf{Pad}\colon \mathsf{Coalg}(T) \to \mathsf{Coalg}(\bar{T})$, which is a fully faithful right-adjoint [SP11].

$$\mathsf{Coalg}(T) \underset{\mathsf{Comp}}{\overset{\mathsf{Pad}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\top}}}} \mathsf{Coalg}(\bar{T})$$

The left adjoint Comp composes the component maps of a multisorted coalgebra in a suitable way as we now explain. For a given $\bar{T}$-coalgebra $(\bar{X}, \bar{\xi})$ with $\bar{\xi} = (\xi_1, \ldots, \xi_n)$ we first define for every subterm T' of T a map

$$c_{T'} \colon X_{f(T')} \to T' X_1$$

by induction as follows: for $T' = (-)$ put $c_{(-)} = \mathsf{id}_{X_1} \colon X_{f(-)} = X_1 \to X_1$, and for $T' = H(G_1, \ldots, G_k)$, $c_{T'}$ is the following map

$$\begin{array}{c} X_{f(T')} = X_{f(H(G_1,\ldots,G_k))} \\ \big\downarrow{\scriptstyle \xi_{f(H(G_1,\ldots,G_k))}} \\ H(X_{f(G_1)}, \ldots, X_{f(G_k)}) \\ \big\downarrow{\scriptstyle H(c_{G_1},\ldots,c_{G_k})} \\ H(G_1 X_1, \ldots, G_k X_1) = T' X_1. \end{array}$$

Then we obtain the T-coalgebra

$$\mathsf{Comp}(\bar{X}, \bar{\xi}) = \big(X_1 \xrightarrow{c_T} TX_{f(T)} = TX_1\big),$$

and on morphisms put $\mathsf{Comp}(h_1, \ldots, h_n) = h_1$. It is not difficult to prove that

$$\mathsf{Comp}(\mathsf{Pad}(X, \xi)) = (X, \xi),$$

see [SP11, Text after Lemma 4.11].

Example 8.4. Given a coalgebra $\xi\colon X \to \mathcal{P}_\mathrm{f}(\mathcal{B}_\mathrm{f} X \times \mathcal{D}(A \times X))$ for T as in Example 8.3, $\mathsf{Pad}(X, \xi)$ is the following $\bar{T}$-coalgebra:

$$\begin{array}{c} (X, \mathcal{B}_\mathrm{f} X \times \mathcal{D}(A \times X), \mathcal{B}_\mathrm{f} X, \mathcal{D}(A \times X), A \times X) \xrightarrow{(\xi, \mathsf{id}, \mathsf{id}, \mathsf{id}, \mathsf{id})} \\ (\mathcal{P}_\mathrm{f}(\mathcal{B}_\mathrm{f} X \times \mathcal{D}(A \times X)), \mathcal{B}_\mathrm{f} X \times \mathcal{D}(A \times X), \mathcal{B}_\mathrm{f} X, \mathcal{D}(A \times X), A \times X) \end{array}$$

Given a $\bar{T}$-coalgebra

$$\bar{\xi}\colon \bar{X} = (X_1, X_2, X_3, X_4, X_5) \xrightarrow{(\xi_1, \xi_2, \xi_3, \xi_4, \xi_5)} (\mathcal{P}_\mathrm{f} X_2, X_3 \times X_4, \mathcal{B}_\mathrm{f} X_1, \mathcal{D} X_5, A \times X_1) = \bar{T}\bar{X}$$

we obtain the following T-coalgebra $\mathsf{Comp}(\bar{X}, \bar{\xi})$ on X_1:

$$X_1 \xrightarrow{\xi_1} \mathcal{P}_\mathrm{f} X_2 \xrightarrow{\mathcal{P}_\mathrm{f} \xi_2} \mathcal{P}_\mathrm{f}(X_3 \times X_4) \xrightarrow{\mathcal{P}_\mathrm{f}(\xi_3 \times \xi_4)} \mathcal{P}_\mathrm{f}(\mathcal{B}_\mathrm{f} X_1 \times \mathcal{D} X_5) \xrightarrow{\mathcal{P}_\mathrm{f}(\mathsf{id} \times \mathcal{D} \xi_5)} \mathcal{P}_\mathrm{f}(\mathcal{B}_\mathrm{f} X_1 \times \mathcal{D}(A \times X_1)).$$

The above coalgebra $\mathsf{Pad}(X,\xi)$ is no longer finite; e.g. $\mathcal{B}_\mathrm{f}(X)$ is infinite for nonempty X. However, one can find a finite $\bar{T}$-coalgebra that conforms to ξ by restricting e.g. the sort $\mathcal{B}_\mathrm{f}(X)$ to those elements of $\mathcal{B}_\mathrm{f}(X)$ that actually appear in ξ.

Note that a functor $H\colon \mathsf{Set}^k \to \mathsf{Set}$ is finitary if and only if for every finite set X and every map $g\colon X \to H(Y_1,\dots,Y_k)$ there exist finite subsets $m_i\colon Y_i' \hookrightarrow Y_i$ such that g factorizes through $H(m_1,\dots,m_k)$:

$$\begin{array}{ccc} X & \xrightarrow{g} & H(Y_1,\dots,Y_k) \\ & \overset{\exists g'}{\searrow} & \uparrow{\scriptstyle H(m_1,\dots,m_k)} \\ & & H(Y_1',\dots,Y_k') \end{array} \tag{8.1}$$

In situations where $Y_i = G(Z_1,\dots,Z_\ell)$ for another finitary functor $G\colon \mathsf{Set}^\ell \to \mathsf{Set}$, this process can be repeated for each of the $m_i\colon Y_i' \to G(Z_1,\dots,Z_\ell)$. Formally:

Construction 8.5. Let $\mathcal{H}$ be a set of finitary functors, let $T\colon \mathsf{Set} \to \mathsf{Set}$ be a functor composed from $\mathcal{H}$ as in Definition 8.2 with flattening $\bar{T}\colon \mathsf{Set}^n \to \mathsf{Set}^n$ (and a bijection f), and let (X,ξ) be a finite T-coalgebra. We construct a $\bar{T}$-coalgebra $\mathsf{Factor}(X,\xi)$ by repeatedly applying the above factorization technique. In this way we obtain finite sets $X_1,\dots,X_n$, for every subterm T' a map $m_{T'}\colon X_{f(T')} \to T'X$, and for every non-leaf subterm $T' = H(G_1,\dots,G_k)$ of T a map $\bar{\xi}_{T'}\colon X_{f(T')} \to H(X_{f(G_1)},\dots,X_{f(G_k)})$. More precisely, we start by putting $m_T := \xi\colon X_1 := X \to TX$ and then proceed recursively down the syntax tree of T. For $T' = (-)$ we put $m_{(-)} = \mathsf{id}_X\colon X_1 = X \to X$, and for a non-leaf subterm $T' = H(G_1,\dots,G_k)$ we apply the above factorization to (the already chosen) $m_{T'}$, i.e. we choose finite subsets $m_{G_i}\colon X_{f(G_i)} \hookrightarrow G_iX$, $i = 1,\dots,k$, and a map $\bar{\xi}_{T'}$ such that the triangle below commutes:

$$\begin{array}{ccc} X & \xrightarrow{m_{T'}} & H(G_1X,\dots,G_kX) \\ & \underset{\bar{\xi}_{T'}}{\searrow} & \uparrow{\scriptstyle H(m_{G_1},\dots,m_{G_n})} \\ & & H(X_{f(G_1)},\dots,X_{f(G_n)}). \end{array} \tag{8.2}$$

For $G_i \neq (-)$, we choose $m_{G_i}\colon X_{f(G_i)} \hookrightarrow G_i(X)$ to be a minimal finite subset admitting such a factorization $\bar{\xi}_{T'}$, and for $G_i = (-)$ we use $m_{(-)} = \mathsf{id}_X$ as defined above. Note that we keep track of duplicate subterms as in Definition 8.2. The i^{th} component of the $\bar{T}$-coalgebra $\mathsf{Factor}(X,\xi)$ is now defined using $f^{-1}(i) = H(G_1,\dots,G_k)$ by

$$X_i = X_{f(H(G_1,\dots,G_k))} \xrightarrow{\bar{\xi}_{H(G_1,\dots,G_k)}} H(X_{f(G_1)},\dots,X_{f(G_k)}) = (\bar{T}(X_1,\dots,X_n))_i.$$

Note that the sets $X_1,\dots,X_n$ are finite, and for $T' \neq T$, $m_{T'}$ is injective.

Proposition 8.6. *For every T-coalgebra (X,ξ) we have*

$$(X,\xi) = \mathsf{Comp}(\mathsf{Factor}(X,\xi)).$$

Moreover,

$$\bar{m} = \big(\mathsf{Factor}(X,\xi) \xrightarrow{(\mathsf{id}_X, m_{f^{-1}(2)}, m_{f^{-1}(3)},\dots,m_{f^{-1}(n)})} \mathsf{Pad}(X,\xi)\big)$$

is a $\bar{T}$-coalgebra morphism, in fact a subcoalgebra inclusion.

Proof. We first verify that $\bar{m}$ is a $\bar{T}$-coalgebra morphism. We do this component-wise and by case distinction.

In the first component, we have $f^{-1}(1) = T = H(G_1, \dots, G_k)$ and

$$\begin{array}{rccc} \text{in } \mathsf{Factor}(X,\xi): & X & \xrightarrow{\bar{\xi}_{H(G_1,\dots,G_k)}} & H(X_{f(G_1)},\dots,X_{f(G_k)}) = \bar{T}(X_1,\dots,X_n)_1 \\ & {\scriptstyle \bar{m}_1 = \mathsf{id}_X}\downarrow & (8.2) & \downarrow{\scriptstyle H(m_{G_1},\dots,m_{G_k}) = (\bar{T}(\bar{m}))_1} \\ \text{in } \mathsf{Pad}(X,\xi): & X & \xrightarrow{\xi = m_T} & H(G_1(X),\dots,G_k(X)) = \bar{T}(X_1,\dots,X_n)_1 \end{array}$$

In all other components we have $f^{-1}(i) = T' = H(G_1, \dots, G_k)$ and

$$\begin{array}{rccc} \text{in } \mathsf{Factor}(X,\xi): & X_{f(H(G_1,\dots,G_k))} & \xrightarrow{\bar{\xi}_{H(G_1,\dots,G_k)}} & H(X_{f(G_1)},\dots,X_{f(G_k)}) = \bar{T}(X_1,\dots,X_n)_i \\ & {\scriptstyle \bar{m}_i = m_{f^{-1}(i)} = m_{T'}}\downarrow & (8.2) & \downarrow{\scriptstyle H(m_{G_1},\dots,m_{G_k}) = (\bar{T}(\bar{m}))_i} \\ \text{in } \mathsf{Pad}(X,\xi): & T'X & \xrightarrow{\mathsf{id}} & H(G_1(X),\dots,G_k(X)). = \bar{T}(X_1,\dots,X_n)_i \end{array}$$

We know that Comp is a functor and that $\mathsf{Comp}(\mathsf{Pad}(X,\xi)) = (X,\xi)$. Then the definition of Comp on morphisms yields $\mathsf{Comp}(\bar{m}) = \bar{m}_1 = \mathsf{id}_X$ and therefore the desired identity.

$$\mathsf{Comp}(\mathsf{Factor}(X,\xi)) \xrightarrow{\mathsf{Comp}(\bar{m})} \mathsf{Comp}(\mathsf{Pad}(X,\xi)) = (X,\xi). \qquad \square$$

Remark 8.7. We do not need Factor to be functorial. In fact, it is functorial if every $H\colon \mathsf{Set}^k \to \mathsf{Set}$ in $\mathcal{H}$ preserves inverse images. However, some of our functors of interest, e.g. $\mathbb{R}^{(-)}$, do not preserve inverse images.

Example 8.8. Consider a finite coalgebra $\xi\colon X \to TX$ for the functor $T = \mathcal{P}_\mathrm{f}(\mathcal{B}_\mathrm{f}(-) \times \mathcal{D}(A \times (-)))$ from Example 8.3. The above Construction 8.5 yields a (multisorted) $\bar{T}$-coalgebra $\mathsf{Factor}(X,\xi)$ with finite carriers (X, X_2, X_3, X_4, X_5), and structure maps

$$\begin{array}{ccccc} \bar{\xi}_T\colon X \to \mathcal{P}_\mathrm{f} X_2 & & \bar{\xi}_{\mathcal{B}_\mathrm{f}(-)\times\mathcal{D}(A\times(-))}\colon X_2 \to X_3 \times X_4 & & \bar{\xi}_{\mathcal{B}_\mathrm{f}(-)}\colon X_3 \to \mathcal{B}_\mathrm{f} X \\ \bar{\xi}_{\mathcal{D}(A\times(-))}\colon X_4 \to \mathcal{D} X_5 & & \bar{\xi}_{A\times(-)}\colon X_5 \to A \times X. & & \end{array}$$

For our purposes it is crucial that we may compute the simple quotient of $\mathsf{Factor}(X,\xi)$ and obtain from its first component the simple quotient of (X,ξ).

Proposition 8.9. *If a $\bar{T}$-coalgebra $(\bar{X},\bar{\xi})$ is simple, then so is the T-coalgebra $\mathsf{Comp}(\bar{X},\bar{\xi})$.*

Proof. Let $q\colon \mathsf{Comp}(\bar{X},\bar{\xi}) \to (Y,\zeta)$; by Proposition 2.8, we need to show that q is monic. From q, we obtain a $\bar{T}$-coalgebra morphism $q'\colon (X,\xi) \to \mathsf{Pad}(Y,\zeta)$ by adjoint transposition; that is,

$$q' = \big((\bar{X},\bar{\xi}) \xrightarrow{\eta_{(\bar{X},\bar{\xi})} = (\mathsf{id}_{X_1}, m_2, \dots, m_n)} \mathsf{Pad}(\mathsf{Comp}(\bar{X},\bar{\xi})) \xrightarrow{\mathsf{Pad}(q) = (q, q_2, \dots, q_n)} \mathsf{Pad}(Y,\zeta)\big).$$

Since $(\bar{X},\bar{\xi})$ is simple, q' is monic in Set^n; in particular, the first component q of q' is monic, as required. $\square$

Corollary 8.10. *If $q\colon \mathsf{Factor}(X,\xi) \twoheadrightarrow (Y,\zeta)$ represents the simple quotient of the $\bar{T}$-coalgebra $\mathsf{Factor}(X,\xi)$, then $\mathsf{Comp}(q)\colon (X,\xi) \to \mathsf{Comp}(Y,\zeta)$ represents the simple quotient of (X,ξ).*

In short, the problem of minimizing single-sorted coalgebras for functors composed in some way from functors H reduces to minimizing multi-sorted coalgebras for the components H.

In the next subsection, we will, in turn, reduce the latter problem to minimizing single-sorted coalgebras, using however coproducts of the components H in lieu of Comp. The benefit of this seemingly roundabout procedure is that refinement interfaces, which fail to combine along functor composition, do propagate along coproducts of functors as we show in Subsection 8.3.

8.2. **De-sorting multisorted coalgebras.** We fix a number n of sorts, and consider coalgebras over $\mathcal{C}^n$. As before, we assume that $\mathcal{C}$, and hence also $\mathcal{C}^n$, fulfils Assumption 2.2. We assume moreover that $H\colon \mathcal{C}^n \to \mathcal{C}^n$ is a mono-preserving functor modelling the transition type of multisorted coalgebras. We now show that under two additional assumptions on $\mathcal{C}$, one can equivalently transform H-coalgebras into single-sorted coalgebras, i.e. coalgebras on $\mathcal{C}$, formed by taking the coproduct of the carriers, a process we refer to as *de-sorting*. Specifically, we need $\mathcal{C}$ to have finite coproducts (implying finite cocompleteness in combination with Assumption 2.2) and to be *extensive* [CLW93]. We begin by taking a closer look at these additional assumptions in the setting of $\mathcal{C}$ and $\mathcal{C}^n$.

Notation 8.11. We have the usual diagonal functor

$$\Delta\colon \mathcal{C} \hookrightarrow \mathcal{C}^n \qquad \Delta(X) = (X, \dots, X).$$

This functor has a left adjoint given by taking coproducts (e.g. [Awo10, p. 225]), which we denote by

$$\coprod\colon \mathcal{C}^n \to \mathcal{C} \qquad \coprod(X_1, \dots, X_n) = X_1 + \ldots + X_n.$$

The unit $\eta_X\colon X \to \Delta\coprod X$ of the adjunction consists of the coproduct injections, and the adjoint transpose of a $\mathcal{C}^n$-morphism $f\colon X \to \Delta Y$, denoted $[f]\colon \coprod X \to Y$, arises by cotupling.

Definition 8.12 [CLW93]**.** A category $\mathcal{C}$ with finite coproducts is called *extensive* if the canonical functor $\mathcal{C}/Y_1 \times \mathcal{C}/Y_2 \to \mathcal{C}/Y_1 + Y_2$ is an equivalence of categories for every pair Y_1, Y_2 of objects.

Remark 8.13. (1) This compact definition can be equivalently rephrased as follows [CLW93, Proposition 2.2]: $\mathcal{C}$ has pullbacks along coproduct injections and a diagram

$$\begin{array}{ccccc} X_1' & \xrightarrow{f_1} & X & \xleftarrow{f_2} & X_2' \\ {\scriptstyle h_1'}\downarrow & & \downarrow{\scriptstyle h} & & \downarrow{\scriptstyle h_2'} \\ Y_1 & \xrightarrow{\mathsf{inl}} & Y_1 + Y_2 & \xleftarrow{\mathsf{inr}} & Y_2 \end{array} \tag{8.3}$$

comprises two pullback squares if and only if the top row is a coproduct diagram.

(2) It easy to see that coproduct injections in an extensive category $\mathcal{C}$ are monomorphic, hence $\eta_X\colon X \rightarrowtail \Delta\coprod X$ is a monomorphism in $\mathcal{C}^n$.

Example 8.14. Categories with Set-like coproducts are extensive, in particular Set itself, the category of partially ordered sets and monotone maps, and the category of nominal sets and equivariant maps. Presheaf categories are extensive, and more generally, so is every Grothendieck topos.

We shall make use of the following equivalent description of extensivity:

Proposition 8.15. *A category $\mathcal{C}$ is extensive if and only if for every $n \geq 0$ we have that a commutative square in $\mathcal{C}^n$ as below is a pullback iff $[f]\colon \coprod X' \to X$ is an isomorphism.*

$$\begin{array}{ccc} X' & \xrightarrow{f} & \Delta X \\ {\scriptstyle h'}\downarrow & & \downarrow{\scriptstyle \Delta h} \\ Y & \xrightarrow{\eta_Y} & \Delta \coprod Y \end{array}$$

Proof. For $n = 0$ and $n = 1$ the above equivalence always holds using that, for $n = 1$, $\eta_Y = \mathsf{id}_Y$. We consider the case $n = 2$. The above diagram then reads:

$$\begin{array}{ccc} (X_1', X_2') & \xrightarrow{(f_1, f_2)} & (X, X) \\ {\scriptstyle (h_1', h_2')}\downarrow & & \downarrow{\scriptstyle (h,h)} \\ (Y_1, Y_2) & \xrightarrow{(\mathsf{inl},\mathsf{inr})} & (Y_1 + Y_2, Y_1 + Y_2) \end{array}$$

This is a pullback in $\mathcal{C}^2$ iff each of its components is a pullback in $\mathcal{C}$, i.e. the two squares in (8.3) are pullbacks. The adjoint transpose $[f]$ is the morphism $[f_1, f_2]\colon X_1' + X_2' \to X$, which is an isormorphism if and only if the top row in (8.3) is a coproduct diagram. Hence, the equivalence in the statement of the proposition for $n = 2$ is equivalent to extensivity. We are done since for $n > 2$ that equivalence clearly follows from that for $n = 2$. □

Our goal in this section is to relate H-coalgebras (in $\mathcal{C}^n$) with $\coprod H\Delta$-coalgebras (in $\mathcal{C}$). This is via two observations:

(1) The obvious functor from H-coalgebras to $H\Delta\coprod$-coalgebras given by

$$(X \xrightarrow{x} HX) \mapsto (X \xrightarrow{x} HX \xrightarrow{H\eta_X} H\Delta\coprod X)$$

preserves and reflects (simple) quotients (Corollary 8.16).

(2) The categories of $\coprod H\Delta$-coalgebras and $H\Delta\coprod$-coalgebras are equivalent (Lemma 8.17).

Since H preserves monomorphisms we have a natural transformation $H\eta\colon H \rightarrowtail H\Delta\coprod$ with monic components. The first translation thus follows from Proposition 2.13:

Corollary 8.16. *If $\mathcal{C}$ is extensive, then a H-coalgebra $\xi\colon X \to HX$ and its induced $H\Delta\coprod$-coalgebra*

$$X \xrightarrow{\xi} HX \xrightarrow{H\eta_X} H\Delta\coprod X$$

have the same quotients and, hence, the same simple quotient.

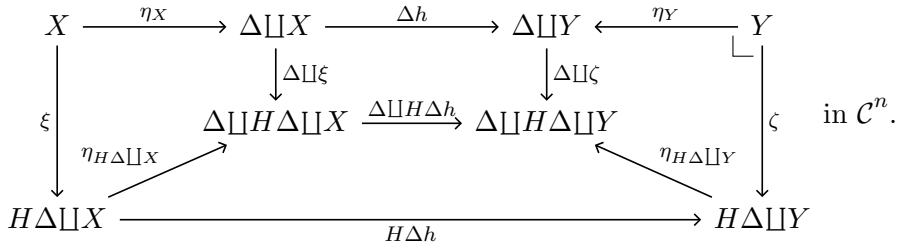
Lemma 8.17. *If $\mathcal{C}$ is extensive, then the lifting $\bar{\coprod}$ of the coproduct functor $\coprod$*

$$\bar{\coprod}\colon \mathsf{Coalg}(H\Delta\coprod) \to \mathsf{Coalg}(\coprod H\Delta), \qquad \bar{\coprod}(X \xrightarrow{\xi} H\Delta\coprod X) = (\coprod X \xrightarrow{\coprod \xi} \coprod H\Delta\coprod X)$$

is an equivalence of categories.

Proof. We have to show that $\bar{\coprod}$ is full, faithful and isomorphism-dense [AHS90]. Faithfulness is immediate from the fact that already $\coprod\colon \mathcal{C}^n \to \mathcal{C}$ is faithful, since coproduct injections are monic (Remark 8.13(2)). To see that $\bar{\coprod}$ is full, let $h\colon \bar{\coprod}(X, \xi) \to \bar{\coprod}(Y, \zeta)$ be a $\coprod H\Delta$-coalgebra

morphism. By naturality of η, we then have a commuting diagram



We have the indicated pullback by extensivity (since $[\eta_Y] = \mathsf{id}$), and thus obtain $h' \colon X \to Y$ such that $\zeta \cdot h' = H\Delta h \cdot \xi$ and $\eta_Y \cdot h' = \Delta h \cdot \eta_X$. Using naturality of η the latter equality yields

$$\Delta\coprod h' \cdot \eta_X = \eta_Y \cdot h' = \Delta h \cdot \eta_X,$$

which implies that $\coprod h' = h$ since η_X is a universal morphism. Now the first equality above states that $h' \colon (X,\xi) \to (Y,\zeta)$ is an $H\Delta\coprod$-coalgebra morphism.

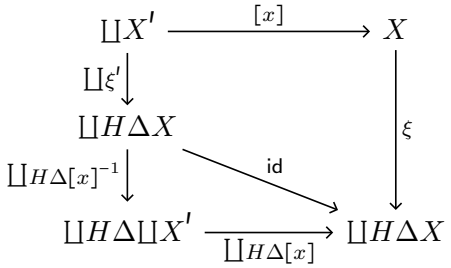
It remains to show that $\bar{\coprod}$ is isomorphism-dense. So let (X,ξ) be a $\coprod H\Delta$-coalgebra. Form the pullback

$$\begin{array}{ccc} X' & \xrightarrow{x} & \Delta X \\ {\scriptstyle \xi'}\downarrow & \lrcorner & \downarrow{\scriptstyle \Delta\xi} \\ H\Delta X & \xrightarrow{\eta_{H\Delta X}} & \Delta\coprod H\Delta X \end{array} \tag{8.4}$$

in $\mathcal{C}^n$. Since $\mathcal{C}$ is extensive, $[x] \colon \coprod X' \to X$ is an isomorphism; we thus have an $H\Delta\coprod$-coalgebra

$$X' \xrightarrow{\xi'} H\Delta X \xrightarrow{H\Delta[x]^{-1}} H\Delta\coprod X'.$$

Applying the adjunction to the square (8.4) shows $\xi \cdot [x] = \coprod\xi'$. It follows that the isomorphism $[x]$ is a coalgebra morphism, and hence an isomorphism in $\mathsf{Coalg}(\coprod H\Delta)$, from $\bar{\coprod}(X', H\Delta[x]^{-1} \cdot \xi')$ to (X,ξ):



□

So the task of computing the simple quotient of a multisorted coalgebra is reduced again to the same problem on ordinary coalgebras in Set. Thus, it remains to check that the arising functor $\Delta H\coprod$ indeed fulfils Assumption 6.9.

8.3. **Coproducts of refinement interfaces.** We have already seen that zippable functors are closed under coproducts (Lemma 5.5). We proceed to show that we can also combine refinement interface along coproducts. Let functors $H_i \colon \mathsf{Set} \to \mathsf{Set}$, $1 \leq i \leq n$ have refinement interfaces with labels A_i and weights W_i, and associated functions $\flat_i, \mathsf{init}_i, w_i, \mathsf{update}_i$. We construct a refinement interface for the coproduct $H = \coprod H_i$, with labels $A = \coprod A_i$ and weights $W = \coprod W_i$, as follows. First define the following helper function, which restricts a multiset of labels to a given sort i:

$$\mathsf{filter}_i \colon \mathcal{B}_\mathrm{f}(\textstyle\coprod_{j=1}^{n} A_j) \to \mathcal{B}_\mathrm{f}(A_i), \qquad \mathsf{filter}_i(f)(a) = f(\mathsf{in}_i(a)).$$

(Note that this differs from the filter function fil_S in (6.1), which filters for a subset of states $S \subseteq X$.) Then we implement the refinement interface for H component-wise as follows (writing $I = \{1, \dots, n\}$):

$$\flat = \Big(\coprod H_i Y \xrightarrow{\coprod \flat_i} \coprod \mathcal{B}_\mathrm{f}(A_i \times Y) \xrightarrow{[\mathcal{B}_\mathrm{f}(\mathsf{in}_i \times Y)]_{i \in I}} \mathcal{B}_\mathrm{f}(\coprod A_i \times Y)\Big)$$

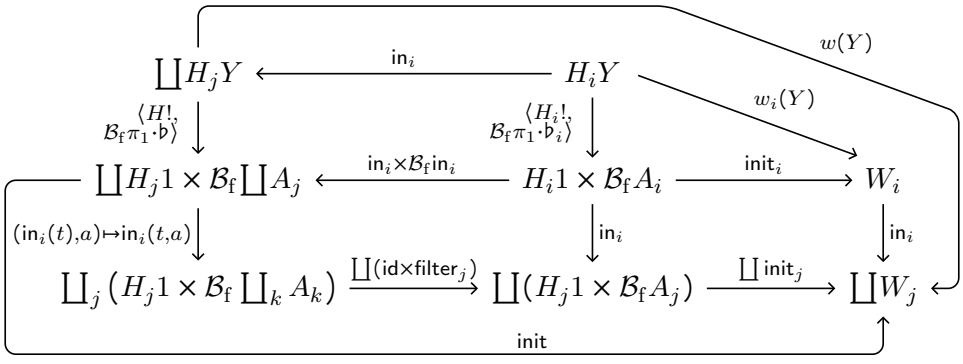
$$w(S) = \Big(\coprod H_i Y \xrightarrow{\coprod w_i(S)} \coprod W_i\Big) \qquad \text{for } S \subseteq Y,$$

$$\mathsf{init} = \Big(\coprod H_i 1 \times \mathcal{B}_\mathrm{f} \coprod A_i \xrightarrow{(\mathsf{in}_i(t),a) \mapsto \mathsf{in}_i(t,a)} \coprod_i \big(H_i 1 \times \mathcal{B}_\mathrm{f} \coprod_j A_j\big) \xrightarrow{\coprod \mathsf{id} \times \mathsf{filter}_i} \coprod (H_i 1 \times \mathcal{B}_\mathrm{f} A_i) \xrightarrow{\coprod \mathsf{init}_i} \coprod W_i\Big)$$

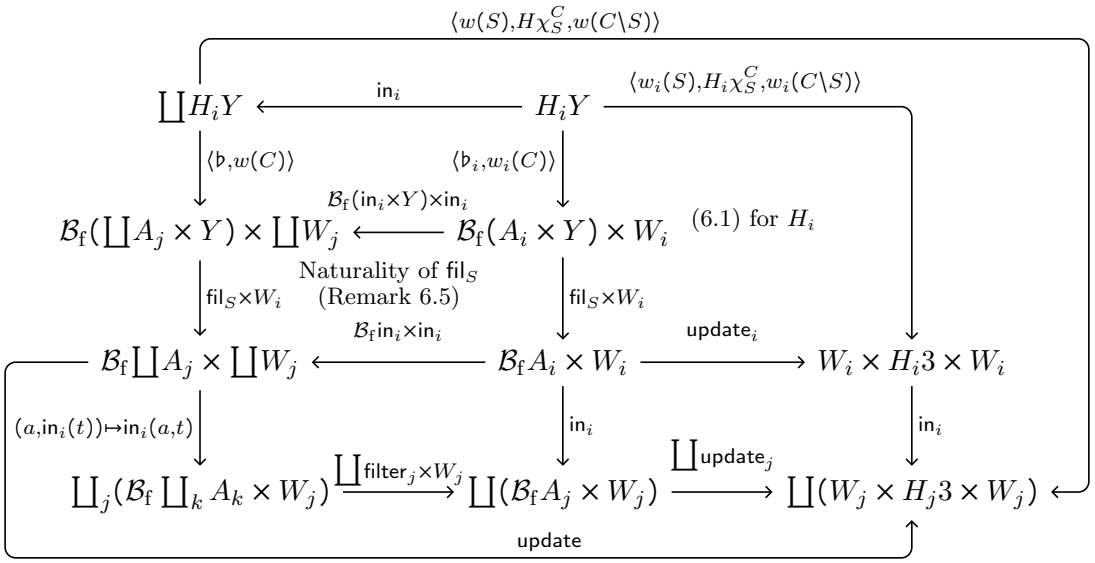
$$\mathsf{update} = \Big(\mathcal{B}_\mathrm{f} \coprod A_i \times \coprod W_i \to \coprod_i \big(\mathcal{B}_\mathrm{f} \coprod_j A_j \times W_i\big) \to \coprod (\mathcal{B}_\mathrm{f} A_i \times W_i) \xrightarrow{\coprod \mathsf{update}_i} \coprod (W_i \times H_i 3 \times W_i) \xrightarrow{[\mathsf{in}_i \times \mathsf{in}_i \times \mathsf{in}_i]_{i \in I}} \coprod W_i \times \coprod H_i 3 \times \coprod W_i\Big)$$

Proposition 8.18. *The data W, A, $\flat$, init, w, and update as constructed above form a refinement interface for $H = \coprod H_i$, and if the interfaces of the H_i fulfil Assumption 6.9, then so does the one of H.*

Proof. For $i = 1, \dots, n$, the following diagram commutes:



Since the $\mathsf{in}_i \colon H_i Y \to \coprod H_i Y$ are jointly epic, the commutativity shows the axiom for init in (6.1). For $S \subseteq C \subseteq Y$ we have the diagram:



Using again that the $\mathsf{in}_i \colon H_i Y \to \coprod H_i Y$ are jointly epic, we see that the claimed refinement interface fulfils the axiom for update in (6.1). If for every i, the interface of H_i fulfils the time constraints from Assumption 6.9, then so does the interface for H: both init and update preprocess the parameters in linear time (via filter), before calling the init_i and update_i of the corresponding H_i. We order $H3$ lexicographically, i.e. $\mathsf{in}_i(x) < \mathsf{in}_j(y)$ iff either $i < j$ or $i = j$ and $x < y$ in $H_i 3$. Comparison in constant time is then clearly inherited. □

Hence, our coalgebraic partition refinement algorithm is modular w.r.t. coproducts. In combination with the results of Sections 8.1 and 8.2, this gives a modular efficient minimization algorithm for multisorted coalgebras, and hence for coalgebras for composite functors.

We proceed to see examples employing the multi-sorted approach, complementing the examples already given for the single-sorted approach (Example 7.18). We build our examples from the functors

$$\mathcal{D}, \mathcal{P}_\mathrm{f}, A \times (-) \colon \mathsf{Set} \to \mathsf{Set} \qquad \times, + \colon \mathsf{Set}^2 \to \mathsf{Set},$$

and in fact many of them appear in work on a coalgebraic hierarchy of probabilistic system types [BSdV03].

Example 8.19. (1) *Labelled transition systems* with an infinite set A of labels. Here we decompose the coalgebraic type functor $\mathcal{P}_\mathrm{f}(A \times (-))$ into $H_1 = \mathcal{P}_\mathrm{f}$ and $H_2 = A \times (-)$. We transform a coalgebra $X \to \mathcal{P}_\mathrm{f}(A \times X)$ (with m edges) into a multisorted (H_1, H_2)-coalgebra; the new sort Y then contains one element per edge. By de-sorting, we finally obtain a single-sorted coalgebra for $\bar{H} = H_1 + H_2$ with $n + m$ states and m edges, leading to a complexity of $\mathcal{O}((n+m) \cdot \log(n+m))$. If $m \geq n$ we thus obtain a run-time in $\mathcal{O}(m \cdot \log m)$, like in [DPP04] but slower than Valmari's $\mathcal{O}(m \cdot \log n)$ [Val09].
For fixed finite A, the running time of our algorithm is in $\mathcal{O}((m+n)\log n)$. Indeed, by finiteness of A, we have $\mathcal{P}_\mathrm{f}(A \times (-)) \cong \mathcal{P}_\mathrm{f}(-)^A$. Then a coalgebra $X \xrightarrow{\xi} \mathcal{P}_\mathrm{f}(A \times X) \cong \mathcal{P}_\mathrm{f}(X)^A$ with $n = |X|$ states and $m = \sum_{x \in X} |\xi(x)|$ edges is transformed into a two-sorted coalgebra

$$\eta_X \colon X \to (A \times X)^A,\ x \mapsto (a \mapsto (a, x)); \qquad \bar{\xi} \colon A \times X \to \mathcal{P}_\mathrm{f}(X),\ (a, x) \mapsto \xi(x)(a).$$

The arising de-sorted system on $X + A \times X$ has $n + |A| \cdot n$ states and $|A| \cdot n + m$ edges, so the simple quotient is found in $\mathcal{O}((m+n) \cdot \log n)$ like in the single-sorted approach (Example 7.18).

(2) As mentioned already, Hopcroft's classical automata minimization [Hop71] is obtained by instantiating our approach to $HX = 2 \times X^A$, with running time $\mathcal{O}(n \cdot \log n)$ for fixed alphabet A. For non-fixed A the best known complexity is in $\mathcal{O}(|A| \cdot n \cdot \log n)$ [Gri73, Knu01]. To obtain the alphabet as part of the input to our algorithm, we consider DFAs as labelled transition systems encoding the letters of the input alphabet as natural numbers. More precisely, given a finite input alphabet A, we choose some injective encoding map $c\colon A \rightarrowtail \mathbb{N}$ and we form the natural transformation with the components

$$m_X\colon 2 \times X^A \rightarrowtail 2 \times \mathcal{P}_\mathrm{f}(\mathbb{N} \times X) \qquad \text{with} \qquad m_X(b, f) = \big(b, \{(c(a), f(a)) \mid a \in A\}\big).$$

Since m_X is clearly monomorphic, we apply Proposition 2.13 to see that minimization of a DFA $\xi\colon X \to 2 \times X^A$ is reduced to minimizing the coalgebra

$$X \xrightarrow{\xi} 2 \times X^A \xrightarrow{m_X} 2 \times \mathcal{P}_\mathrm{f}(\mathbb{N} \times X).$$

Further, we decompose the type functor into $H_1 = 2 \times \mathcal{P}_\mathrm{f}$ and $H_2 = \mathbb{N} \times (-)$. An automaton ξ for a finite input alphabet A is then represented by the two-sorted system

$$\xi\colon X \to 2 \times \mathcal{P}_\mathrm{f}(A \times X) \qquad c \times X\colon A \times X \to \mathbb{N} \times X$$

With $|X| = n$, this system has $n + |A| \cdot n$ states and $|A| \cdot n + |A| \cdot n$ edges. Thus, our algorithm runs in time

$$\mathcal{O}((|A| \cdot n) \cdot \log(|A| \cdot n)) = \mathcal{O}(|A| \cdot n \cdot \log n + |A| \cdot n \cdot \log |A|),$$

i.e. as fast as the above-mentioned best known algorithms except on automata with more alphabet letters than states.

(3) Coalgebras for the functor $HX = \mathcal{D}X + \mathcal{P}_\mathrm{f}(A \times X)$ are *alternating systems* [Han94]. The functor H is flattened to the multi-sorted functor

$$\bar{H}(X_1, X_2, X_3, X_4) = (X_2 + X_3, \mathcal{D}X_1, \mathcal{P}_\mathrm{f}X_4, A \times X_1)$$

on Set^4, which is then de-sorted to obtain the Set-functor

$$\bigsqcup \bar{H} \Delta X = (X + X) + \mathcal{D}X + \mathcal{P}_\mathrm{f}X + A \times X$$

which has a refinement interface as given by Proposition 8.18. Given an H-coalgebra with n states and m_d edges of type $\mathcal{D}$ and m_p edges of type $\mathcal{P}_\mathrm{f}(A \times (-))$, the induced $\bigsqcup \bar{H} \Delta$-coalgebra has $n + m_p$ states and $n + m_d + m_p + m_p$ edges, and is minimized under bisimilarity in time $\mathcal{O}((n + m_d + m_p) \cdot \log(n + m_p))$.

Other probabilistic system types [BSdV03] are handled similarly, where one only needs to take care of estimating the number of states in the intermediate sorts as in the treatment above. We discuss two further examples explicitly, simple and general Segala systems.

(4) For a simple Segala system considered as a coalgebra $\xi\colon X \to \mathcal{P}_\mathrm{f}(A \times \mathcal{D}X)$ there are partition refinement algorithms by Baier, Engelen, Majster-Cederbaum [BEM00] and by Groote, Verduzco, and de Vink [GVdV18]. For the complexity analysis, define the number of states and edges respectively as

$$n = |X|, \qquad m_p = \textstyle\sum_{x \in X} |\xi(x)|.$$

The arising multi-sorted coalgebra consists of maps

$$p\colon X \to \mathcal{P}_{\mathrm{f}} Y \qquad a\colon Y \to A \times Z \qquad d\colon Z \to \mathcal{D}X.$$

In the coalgebra ξ there is one distribution per non-deterministic edge, hence $|Y| = m_p = |Z|$. The non-deterministic map p has m_p edges by construction, and the deterministic map a has $|Y| = m_p$ edges. Let m_d denote the number of edges needed to encode d; then $m_d \leq n \cdot m_p$. We thus have $n + 2 \cdot m_p$ states and $2 \cdot m_p + m_d$ edges, so our algorithm runs in time $\mathcal{O}((n + m_p + m_d) \cdot \log(n + m_p))$. For every $z \in Z$, $d(z)$ is a non-empty distribution, and so under the assumption that there is at least one non-deterministic edge per state $x \in X$, we have $m_d \geq m_p \geq n$, simplifying the complexity to $\mathcal{O}(m_d \cdot \log m_p)$. In independent work, Groote et al. [GVdV18] consider Z as part of the input, and design and implement an algorithm of time complexity $\mathcal{O}((m_p + m_d) \cdot \log |Z| + m_d \cdot \log n)$, which simplifies to the same complexity $\mathcal{O}(m_d \cdot \log m_p)$ for $|Z| = m_p$ and $m_d \geq m_p \geq n$. This is more fine-grained than the complexity $\mathcal{O}((n \cdot m_p) \cdot \log(n + m_p)) = \mathcal{O}((n \cdot m_p) \cdot \log(n \cdot m_p))$ of [BEM00], and indeed leads to a faster run-time in the (presumably wide-spread) case that probabilistic transitions are sparse, i.e. if m_d is substantially below $n \cdot m_p$.

(5) For a general Segala system $\xi\colon X \to \mathcal{P}_{\mathrm{f}}(\mathcal{D}(A \times X))$ one has a similar factorization:

$$p\colon X \to \mathcal{P}_{\mathrm{f}} Y \qquad d\colon Y \to \mathcal{D}Z \qquad z\colon Z \to A \times X$$

So for $n = |X|$ states, m_p non-deterministic edges, and m_d probabilistic edges, the multisorted system has $n + |Y| + |Z| = n + m_p + m_d$ states and $m_p + m_d + m_d$ edges. Since $d(y)$ is non-empty for all $y \in Y$, $m_d \geq m_p$, and so the generic partition refinement has a run-time of $\mathcal{O}(m_d \cdot \log(n + m_d))$.

Summing up the last three examples, on simple Segala systems we obtain faster run-time than the best previous algorithm [BEM00] (with similar results obtained independently by Groote et al. [GVdV18]), and we obtain, to our best knowledge, the first similarly efficient partition refinement algorithms for alternating systems (Example 8.19(3)) and general Segala systems (Example 8.19(5)).

9. Conclusions and Further Work

We have presented a generic algorithm that quotients coalgebras by behavioural equivalence. We have started from a category-theoretic procedure that works for every mono-preserving functor on a category with image factorizations, and have then developed an improved algorithm for *zippable* endofunctors on Set. Provided the given type functor can be equipped with an efficient implementation of a *refinement interface*, we have finally arrived at a concrete procedure that runs in time $\mathcal{O}((m+n)\log n)$ where m is the number of edges and n the number of nodes in a graph-based representation of the input coalgebra. We have shown that this instantiates to (minor variants of) several known efficient partition refinement algorithms: the classical Hopcroft algorithm [Hop71] for minimization of DFAs, the Paige-Tarjan algorithm for unlabelled transition systems [PT87], Valmari and Franceschinis's lumping algorithm for weighted transition systems [VF10], and the 1-dimensional Weisfeiler Lehman Algorithm [CFI92, Wei76, SSvL$^+$11]. Moreover, we have presented a generic method to apply the algorithm to mixed system types. As an instance, we obtain an algorithm for simple Segala systems that allows for a more fine-grained analysis of asymptotic run-time than previous algorithms [BEM00], and matches the run-time of a recent algorithm described independently by Groote et al. [GVdV18].

Further instances can be covered by relaxing the time complexity assumptions on the refinement interfaces [DMSW19], which allows covering monoid-valued functors $M^{(-)}$ also for non-cancellative monoids M; by our compositionality methods, we obtain in particular efficient partition refinement algorithm for M-weighted tree automata, which improves the run-time of a previous algorithm by Högberg, Maletti, and May [HMM07].

In further work [DMSW19], we describe and evaluate a generic implementation CoPaR of the generic algorithm. The implementation supports unrestricted combination of functors with refinement interfaces (cf. Section 8) and implements all the functors from Section 7.

It remains open whether our approach can be extended to, e.g. the monotone neighbourhood functor, which is not itself zippable (see Example 5.9) and also does not have an obvious factorization into zippable functors. We do expect that our algorithm applies beyond weighted systems. For example, it should be relatively straightforward to extend our algorithm to nominal systems, i.e. coalgebras for functors on the category of nominal sets and equivariant maps. Of course, precise complexity bounds will then depend on the representation of nominal sets.

References

[ABH+12] Jirí Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius, and Alexandra Silva. A coalgebraic perspective on minimization and determinization. volume 7213 of *LNCS*, pages 58–73. Springer, 2012.

[Adá05] Jiří Adámek. Introduction to coalgebra. *Theory Appl. Categ.*, 14:157–199, 2005.

[AHS90] Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and Concrete Categories*. Wiley Interscience, 1990.

[AM89] Peter Aczel and Nax Mendler. A final coalgebra theorem. In *Proc. Category Theory and Computer Science (CTCS)*, volume 389 of *Lecture Notes Comput. Sci.*, pages 357–365. Springer, 1989.

[AR94] Jiří Adámek and Jiří Rosický. *Locally presentable and accessible categories*. Cambridge University Press, 1994.

[AT90] Jiří Adámek and Věra Trnková. *Automata and Algebras in Categories*. Kluwer, 1990.

[Awo10] Steve Awodey. *Category Theory*. Oxford Logic Guides. OUP Oxford, 2010.

[Bac86] Roland Backhouse. *Program Construction and Verification*. Prentice-Hall, 1986.

[BBG17] Christoph Berkholz, Paul S. Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory Comput. Syst.*, 60(4):581–614, 2017.

[BEM00] Christel Baier, Bettina Engelen, and Mila Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *J. Comput. Syst. Sci.*, 60:187–231, 2000.

[BGK+19] Olav Bunte, Jan Friso Groote, Jeroen J. A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A. C. Willemse. The mcrl2 toolset for analysing concurrent systems - improvements in expressivity and usability. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2019, Part II*, pages 21–39, 2019.

[BO05] Stefan Blom and Simona Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. *STTT*, 7(1):74–86, 2005.

[BSdV03] Falk Bartels, Ana Sokolova, and Erik de Vink. A hierarchy of probabilistic system types. In *Coagebraic Methods in Computer Science, CMCS 2003*, volume 82 of *ENTCS*, pages 57 – 75. Elsevier, 2003.

[Buc08] Peter Buchholz. Bisimulation relations for weighted automata. *Theoret. Comput. Sci.*, 393:109–123, 2008.

[CFI92] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, dec 1992.

[CLW93] Aurelio Carboni, Steve Lack, and Robert F. C. Walters. Introduction to extensive and distributive categories. *J. Pure Appl. Algebra*, 84:145–158, 1993.

[CS02] Stefano Cattani and Roberto Segala. Decision algorithms for probabilistic bisimulation. In *Concurrency Theory, CONCUR 2002*, volume 2421 of *LNCS*, pages 371–385. Springer, 2002.

[DEP02] Josee Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. *Inf. Comput.*, 179(2):163–193, 2002.

[DHS03] Salem Derisavi, Holger Hermanns, and William Sanders. Optimal state-space lumping in markov chains. *Inf. Process. Lett.*, 87(6):309–315, 2003.

[DMSW17] Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Efficient coalgebraic partition refinement. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl, 2017.

[DMSW19] Hans-Peter Deifel, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Generic partition refinement and weighted tree automata. In *Formal Methods, FM 2019*, LNCS. Springer, 2019. To appear. Preprint available on arXiv at `https://arxiv.org/abs/1811.08850`.

[DPP04] Agostino Dovier, Carla Piazza, and Alberto Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 311(1-3):221–256, 2004.

[FV02] Kathi Fisler and Moshe Vardi. Bisimulation minimization and symbolic model checking. *Formal Methods in System Design*, 21(1):39–78, 2002.

[GJKW17] Jan Friso Groote, David N. Jansen, Jeroen J. A. Keiren, and Anton Wijs. An $O(m \log n)$ algorithm for computing stuttering equivalence and branching bisimulation. *ACM Trans. Comput. Log.*, 18(2):13:1–13:34, 2017.

[Gri73] David Gries. Describing an algorithm by Hopcroft. *Acta Informatica*, 2:97–109, 1973.

[GS01] Heinz-Peter Gumm and Tobias Schröder. Monoid-labelled transition systems. In *Coalgebraic Methods in Computer Science, CMCS 2001*, volume 44 of *ENTCS*, pages 185–204, 2001.

[GVdV18] Jan Friso Groote, Jao Rivera Verduzco, and Erik P. de Vink. An efficient algorithm to determine probabilistic bisimulation. *Algorithms*, 11(9):131, 2018.

[Han94] Hans Hansson. *Time and Probability in Formal Design of Distributed Systems*. Elsevier, 1994.

[HMM07] Johanna Högberg, Andreas Maletti, and Jonathan May. Bisimulation minimisation for weighted tree automata. In *Developments in Language Theory, DLT 2007*, volume 4588 of *LNCS*, pages 229–241. Springer, 2007.

[HMM09] Johanna Högberg, Andreas Maletti, and Jonathan May. Backward and forward bisimulation minimization of tree automata. *Theor. Comput. Sci.*, 410:3539–3552, 2009.

[Hop71] John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

[HT92] Dung Huynh and Lu Tian. On some equivalence relations for probabilistic processes. *Fund. Inform.*, 17:211–234, 1992.

[Ihr03] Thomas Ihringer. *Algemeine Algebra. Mit einem Anhang über Universelle Coalgebra von H. P. Gumm*, volume 10 of *Berliner Studienreihe zur Mathematik*. Heldermann Verlag, 2003.

[Jac17] Bart Jacobs. *Introduction to Coalgebras: Towards Mathematics of States and Observations*. Cambridge University Press, 2017.

[JR97] Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bull. EATCS*, 62:222–259, 1997.

[KK14] Barbara König and Sebastian Küpper. Generic partition refinement algorithms for coalgebras and an instantiation to weighted automata. In *Theoretical Computer Science, IFIP TCS 2014*, volume 8705 of *LNCS*, pages 311–325. Springer, 2014.

[KKZJ07] Joost-Pieter Katoen, Tim Kemna, Ivan Zapreev, and David Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2007*, volume 4424 of *LNCS*, pages 87–101. Springer, 2007.

[Kli09] Bartek Klin. Structural operational semantics for weighted transition systems. In Jens Palsberg, editor, *Semantics and Algebraic Specification: Essays Dedicated to Peter D. Mosses on the Occasion of His 60th Birthday*, volume 5700 of *LNCS*, pages 121–139. Springer, 2009.

[Knu01] Timo Knuutila. Re-describing an algorithm by Hopcroft. *Theor. Comput. Sci.*, 250:333 – 363, 2001.

[KS83] Paris C. Kanellakis and Scott A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 228–240, New York, NY, USA, 1983. ACM.

[KS90] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990.

[LS91] Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94:1–28, 1991.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[MM92] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic*. Springer New York, 1992.

[MPW19] Stefan Milius, Dirk Pattinson, and Thorsten Wißmann. A new foundation for finitary corecursion and iterative algebras, 2019. Submitted, preprint available on `https://arxiv.org/abs/1802.08070`.

[Par81] David Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science, 5th GI-Conference*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[RT08] Francesco Ranzato and Francesco Tapparo. Generalizing the Paige-Tarjan algorithm by abstract interpretation. *Inf. Comput.*, 206:620–651, 2008.

[Rut00] Jan Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249:3–80, 2000.

[Seg95] Roberto Segala. *Modelling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

[SP11] Lutz Schröder and Dirk Pattinson. Modular algorithms for heterogeneous modal logics via multi-sorted coalgebra. *Math. Struct. Comput. Sci.*, 21(2):235–266, 2011.

[SSvL+11] Nino Shervashidze, Pascal Schweitzer, Erik van Leeuwen, Kurt Mehlhorn, and Karsten Borgwardt. Weisfeiler-Lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.

[Val09] Antti Valmari. Bisimilarity minimization in $\mathcal{O}(m \log n)$ time. In *Applications and Theory of Petri Nets, PETRI NETS 2009*, volume 5606 of *LNCS*, pages 123–142. Springer, 2009.

[vB77] Johann van Benthem. *Modal Correspondence Theory*. PhD thesis, Universiteit van Amsterdam, 1977.

[vdMZ07] Ron van der Meyden and Chenyi Zhang. Algorithmic verification of noninterference properties. In *Views on Designing Complex Architectures, VODCA 2006*, volume 168 of *ENTCS*, pages 61–75. Elsevier, 2007.

[VF10] Antti Valmari and Giuliana Franceschinis. Simple $\mathcal{O}(m \log n)$ time Markov chain lumping. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2010*, volume 6015 of *LNCS*, pages 38–52. Springer, 2010.

[Wei76] Boris Weisfeiler. *On Construction and Identification of Graphs*. Springer, 1976.

[Wor05] James Worrell. On the final sequence of a finitary set functor. *Theor. Comput. Sci.*, 338:184–199, 2005.

[ZHEJ08] Lijun Zhang, Holger Hermanns, Friedrich Eisenbrand, and David Jansen. Flow Faster: Efficient decision algorithms for probabilistic simulations. *Log. Meth. Comput. Sci.*, 4(4), 2008.

This work is licensed under the Creative Commons Attribution License. To view a copy of this license, visit `https://creativecommons.org/licenses/by/4.0/` or send a letter to Creative Commons, 171 Second St, Suite 300, San Francisco, CA 94105, USA, or Eisenacher Strasse 2, 10777 Berlin, Germany