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, a task arising in different contexts. Coalgebraic generality allows us to not only cover classical relational systems and various forms of weighted systems but way can combine existing system types in various ways. 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 results and the possibilities of combining system types is a toolbox for efficient partition refinement algorithms. Instances of our generic algorithm thus match the runtime of the best known algorithms for unlabelled transition systems, Markov chains, and deterministic automata (with fixed alphabets), and improve the best known algorithms for Segala systems.

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, called *minimization under bisimilarity*.

The computation of reachable states is usually accomplished by a straightforward search. 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] and weighted automata [Buc08]. In fact, 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

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

Preprint submitted to
Logical Methods in Computer Science

balance [FV02], in probabilistic model checking, minimization under strong bisimilarity does lead to substantial efficiency gains [KKZJ07].

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. In the relational setting, Kanellakis and Smolka [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, thus achieving better average-case behaviour but the same worst-case behaviour [DPP04]. Probabilistic minimization has undergone a similarly dynamic development [BEM00, CS02, ZHEJ08], 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.

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). We achieve this by methods of *universal coalgebra* [Rut00]. 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 bring out and 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. [KK14]), the other routine is "select the smaller half", a trick going back to Hopcroft [Hop71] that will lead to the logarithmic factor in the complexity analysis later.

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 Section 6, this incremental computation of the partitions is given as an algorithm in pseudocode, with the "select the smaller half" routine hard-wired. To this end, we make our algorithm parametric in an abstract *refinement interface* of the type functor, which encapsulates how the functor interacts with the specific select routine. We show that if the 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]), as well as Hopcroft's classical automata minimization algorithm [Hop71] for a fixed alphabet A and $m = n \cdot |A|$.

Section 7 is devoted to modularity and explains how to handle combinations of system types. 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 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. Then we prove that for every multisorted functor $H\colon \mathsf{Set}^m \to \mathsf{Set}^m$, we have a Set-functor $\bigsqcup H\Delta\colon \mathsf{Set} \to \mathsf{Set}$, where Δ is diagonal and $\bigsqcup$ takes coproducts, and a transformation from multisorted H-coalgebras to singlesorted $\bigsqcup H\Delta$-coalgebras that preserves the number of states and preserves and reflects bisimilarity minimization. This yields a reduction from bisimilarity minimization of T-coalgebras to minimization of $\bigsqcup \bar{T}\Delta$-coalgebras. The latter problem is solved by the algorithm from Section 6, because we prove that if T is built from functors fulfilling our assumption, then $\bigsqcup \bar{T}\Delta$ 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 7.17).

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 nonzero probabilistic edges enter into the time bound). We also obtain efficient minimization algorithms for general Segala systems and alternating systems [Han94].

This paper is an extended and completely reworked version of the conference paper [DMSW17]. Besides providing detailed proofs of all our results, we have included the new Section 7 showing that modularity is achieved without any adjustment of our algorithm.

2. Preliminaries

It is advisable for readers to be familiar with basic category theory [AHS90]. However, the results are understandable when 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 $! : A \to 1$. 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, i.e. injections in Set. We write $q\colon A \twoheadrightarrow B$ for regular epimorphisms, this means 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 relating $f(r)$ and $g(r)$ for all $r \in R$. Here, one can think of R as a set of witnesses of the pairs $f(r), g(r)$ generating that equivalence (note that there might be several witnesses for a pair provided by f, g). Hence, we will often denote the coequalizer of $f, g\colon R \rightrightarrows A$ by $\kappa_R\colon A \twoheadrightarrow A/R$ and this 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 $q : A \twoheadrightarrow A/{\sim}$, and there is a bijection between equivalence relations and and partitions on A. In order to obtain a similar bijection for more general categories than Set we work under the following

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 set of objects $\{1, \dots, n\}$.

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

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

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

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 $A/\mathsf{ker}\, f \rightarrowtail A$; together with $[-]_f$, this is the factorization of f into a regular epimorphism and a monomorphism.

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 $f = m \cdot e$

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

where m is a mono and e is a regular epimorphism, specifically the coequalizer of the kernel $\pi_1, \pi_2\colon \mathsf{ker}\, f \rightrightarrows A$. Its codomain, $\mathsf{Im}(f)$ is called the *image* of f. In every category, we have the *diagonal property* for monomorphisms m and regular epimorphisms e: Whenever $f \cdot e = m \cdot g$ then there exists a diagonal d such that $d \cdot e = g$ and $m \cdot d = f$.

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, there are partial orders on both kernels and quotients such that the above bijection is monotone. In fact, 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 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* then a relation $\langle p_1', p_2'\rangle\colon R' \rightarrowtail A \times B$ if there exists a (necessarily monic) $m\colon R \rightarrowtail R'$ such that $p_i' \cdot m = p_i$, for $i = 1, 2$. We use intersection $\cap$ and union $\cup$ of kernels for meets and joins in the inclusion ordering on *relations* (not equivalence relations or kernels) on A; in this notation,

$$\mathsf{ker}\langle f, g\rangle = \mathsf{ker}\, f \cap \mathsf{ker}\, g. \tag{2.1}$$

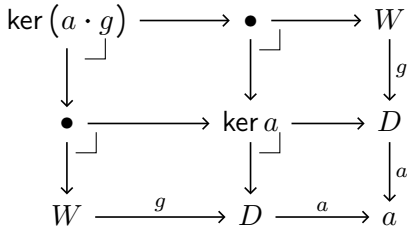
Similarly, a quotient represented by $q_1\colon A \twoheadrightarrow B_1$ is finer than another one represented by $q_2\colon A \twoheadrightarrow B_2$ if there exists a (necessarily regular epic) morphism $b\colon B_1 \twoheadrightarrow B_2$ with $q_2 = b \cdot q_1$.

We will use a number of simple observations on kernels that are familiar when instantiating them to Set:

Remark 2.4. (1) For every $f\colon X \to Y$ and $g\colon Y \to Z$, $\mathsf{ker}(f)$ is finer than $\mathsf{ker}(g \cdot f)$.
(2) For every $f\colon X \to Y$ and every mono $m\colon Y \rightarrowtail Z$, $\mathsf{ker}(m \cdot f) = \mathsf{ker}\, f$.
(3) For every $f\colon X \to Y$ and regular epi $q\colon X \twoheadrightarrow Z$, $\mathsf{ker}(f) = \mathsf{ker}(q)$ 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 $\mathsf{ker}(e) = \mathsf{ker}(f)$ 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 $e\colon X \twoheadrightarrow Y$, $\mathsf{ker}(e)$ is finer than $\mathsf{ker}(f)$ iff there exists some morphism $g\colon Y \to Z$ such that $g \cdot e = f$.

    To see this let $p, q\colon \mathsf{ker}(e) \rightrightarrows X$ be the kernel pair of e. If $\mathsf{ker}(e)$ is finer than $\mathsf{ker}(f)$, 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 $\mathsf{ker}(a\colon D \to A) = \mathsf{ker}(b\colon D \to B)$ then $\mathsf{ker}(a \cdot g) = \mathsf{ker}(b \cdot g)$, for every $g\colon W \to D$.

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

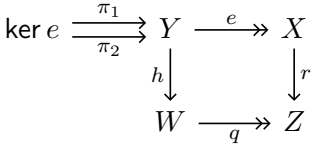


So if $\mathsf{ker}\, a = \mathsf{ker}\, b$, then $\mathsf{ker}(a \cdot g) = \mathsf{ker}(b \cdot g)$.

Even though only existence of coequalizer 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



is a pushout. Uniqueness of any mediating morphism is clear since q is epic; so we need to show existence. Let $W \xrightarrow{g} U \xleftarrow{f} X$ be such that $fe = gh$. 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 $kq = g$. It remains to check that $k \cdot r = f$. Now $k \cdot r \cdot e = k \cdot q \cdot h = g \cdot h = f \cdot e$, which implies the claim because e is epic. □

2.2. **Coalgebra.** We now briefly recall basic notions from the theory of coalgebras. For introductory texts, see [Rut00, JR97, Adá05, Jac17]. Given an endofunctor $H\colon \mathcal{C} \to \mathcal{C}$, a *coalgebra* is a pair (C, c) where C is an object of $\mathcal{C}$ called the *carrier* and thought of as an object of *states*, and $c\colon C \to HC$ 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 $\mathcal{P}$). Explicitly, a coalgebra $c\colon X \to HX$ assigns to each state x a set $c(x) \in \mathcal{P}(A \times X)$, and this represents the transition structure at x: x has an a-transition to y iff $(a, y) \in c(x)$. In concrete examples, we restrict to the finite powerset $\mathcal{P}_\mathrm{f} X = \{S \in \mathcal{P}X \mid S \text{ finite}\}$, 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 by

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

and for a map $h\colon X \to Y$ by

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

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

(3) The finite powerset functor $\mathcal{P}_\mathrm{f}$ is a monoid-valued functor 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 any label $a \in A$ a state non-deterministically proceeds to one of a finite number of possible probability distributions over states; and 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.

A *coalgebra morphism* from a coalgebra (C, c) to a coalgebra (D, d) is a morphism $h\colon C \to D$ such that $d \cdot h = Hh \cdot c$; 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:

$$\begin{array}{ccc} C & \xrightarrow{c} & HC \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle Hh} \\ D & \xrightarrow{d} & HD \end{array}$$

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

A *subcoalgebra* of a coalgebra (C, c) is represented by a coalgebra morphism $m\colon (D, d) \rightarrowtail (C, c)$ such that m is a monomorphism in $\mathcal{C}$. Likewise, a *quotient* of a coalgebra (C, c) is represented by a coalgebra morphism $q\colon (C, c) \twoheadrightarrow (D, d)$ carried by a regular epimorphism q of $\mathcal{C}$. If H preserves monomorphisms, then the image factorization structure on $\mathcal{C}$ lifts to coalgebras.

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

Proposition 2.8. *A coalgebra (C, c) is simple iff every coalgebra morphism with domain (C, c) is carried by a monomorphism.*

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. *The simple quotient of a coalgebra is unique (up to isomorphism). Concretely, let (C, c) be a coalgebra, and let $e_i : (C, c) \twoheadrightarrow (D_i, d_i)$, $i = 1, 2$, 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 $D_1 \xrightarrow{f_1} E \xleftarrow{f_2} D_2$ of $D_1 \xleftarrow{e_1} C \xrightarrow{e_2} D_2$ in $\mathsf{Coalg}(H)$. Since regular epimorphisms are generally stable under pushouts, f_1 and f_2 are regular epimorphisms, hence isomorphisms because D_1 and D_2 are simple; this proves the claim. □

For $\mathcal{C} = \mathsf{Set}$, two elements $x \in C$ and $y \in D$ of coalgebras (C, c) and (D, d) are called *behaviourally equivalent* (written $x \sim y$) if they can be merged by coalgebra morphisms: $x \sim y$ iff there exists a coalgebra (E, e) and coalgebra morphisms $f : (C, c) \to (E, e)$, $g : (D, d) \to (E, e)$ such that $f(x) = g(y)$. For general categories $\mathcal{C}$, we consider (generalized) elements, i.e. two morphisms $x : X \to C$ and $y : Y \to D$, and we have $x \sim y$ iff there are f and g as above with $fx = gy$.

Under Assumption 2.2, any two behaviourally equivalent elements (of the same coalgebra) can be identified under a regular quotient. Hence, the simple quotient of a coalgebra is its quotient modulo behavioural equivalence. In our main examples, this means that we minimize w.r.t. standard bisimilarity-type equivalences.

Example 2.10. 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 2.11. A *final coalgebra* is a terminal object in the category of coalgebras, i.e. a coalgebra (T, t) such that every coalgebra (D, d) has a unique coalgebra morphism into (T, t). 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. If (T, t) is a final coalgebra and H preserves monos, then the simple quotient of a coalgebra (D, d) is the image of (D, d) under the unique morphism into (T, t); in particular, in this case every coalgebra has a simple quotient.

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.

So before stating the abstract Algorithm 4.5, we will give an informal description of a partition refinement algorithm in this section, which will make it clear which parts of partition refinement algorithms are generic and which parts are specific to a particular transition type.

Given a system with a set of states X 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 partitions of X, 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; and in the case of labelled transition systems, the initial partition separates deadlock states from states with successors.

Algorithm 3.1 (Informal Partition Refinement)**.** Given a system on X and initial partitions X/P and X/Q; while P is properly finer than Q, iterate the following:

(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 $q\colon X \to \{S, C \setminus S, X \setminus C\}$.
(2) Refine X/Q with respect to this quotient by splitting C into two blocks S and $C \setminus S$; this is just the (block-wise) intersection of the partitions X/Q and $\{S, C \setminus S, X \setminus C\}$.
(3) Choose X/P as coarse as possible such that two states are identified in X/P if their transition structure is the same up to Q. Sometimes, this is referred to as refining X/P in a way such that the transition structure is *stable* w.r.t. X/Q.

Since X/Q is refined using information from X/P with each iteration, the partition X/P is finer than X/Q invariantly.

Now note that Steps (1) and (2) are independent of the given transition type, because they perform 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.

4. A Categorical Algorithm for Behavioural Equivalence

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

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

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

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

- Analyse $X \xrightarrow{\xi} HX \xrightarrow{Hq} HX/Q$ 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.
- Use parts of this information to refine q.

Here q corresponds to the partition X/Q and q together with X/Q and $Hq\dot{\xi}$ to the finer partition X/P in Algorithm 3.1 from the previous section. The subblock selection will be encapsulated at the present level of generality in a routine select, assumed as a parameter of our algorithm:

Definition 4.3. A select routine is an operation that receives a chain of two regular epis $X \xrightarrow{y}\!\!\!\!\rightarrow Y \xrightarrow{z}\!\!\!\!\rightarrow Z$ and returns some morphism $k : Y \to K$. We call Y the *subblocks* and Z the *compound blocks*.

In our algorithm, y represents the canonical quotient $X \twoheadrightarrow X/P$ and z represents the canonical $X/P \twoheadrightarrow X/Q$, given by the invariant that P is finer than Q. The idea is that the morphism k selects some of the information contained in Y. For example, in the Paige-Tarjan algorithm it models the selection of only a single compound block to be split in two, refining Z, which then induces further refinement of Y (as opposed to splitting all compound blocks simultaneous).

Example 4.4. (1) In Hopcroft's algorithm [Hop71], and in fact 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. We use the same strategy for our algorithm whenever $\mathcal{C} = \mathsf{Set}$ (see Section 6) with the following select routine. Let $S \in Y$ such that $2 \cdot |y^{-1}[\{S\}]| \leq |(zy)^{-1}[\{z(S)\}]|$. Here, $z(S)$ is the compound block containing the subblock S.

Then $\mathsf{select}(z, y)$ is $k : Y \to 3$ given by

$$k(x) = \begin{cases} 2 & \text{if } x = S \\ 1 & \text{if } z(x) = z(S) \\ 0 & \text{otherwise.} \end{cases}$$

If there is no such $S \in Y$, then z is bijective, i.e., there is no compound block from Z that needs to be refined. In this case, k does not matter and we simply put $k = ! : Y \to 1$.

In general, for any subsets $S \subseteq C \subseteq X$, define the following characteristic function:

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

This three-valued version is essentially $\langle \chi_S, \chi_C \rangle \colon X \to 2 \times 2$ without the impossible case of $x \in S \setminus C$. With k, S, and C as above we have

$$k = \chi_{\{S\}}^{[S]_z} \quad \text{and} \quad k \cdot y = \chi_S^C.$$

(2) One obvious choice for k is the identity on Y, so that *all* of the information present in Y is used for further refinement. We will discuss this in Remark 4.11.

(3) Two other, trivial, choices are $k = ! : Y \to 1$ and $k = z$. Since both of these choices provide no extra information, this will leave the partitions unchanged, see the proof of Theorem 4.14.

Given a select routine, the most general form of our partition refinement works as follows.

Algorithm 4.5. Given a coalgebra $\xi : X \to HX$, we successively refine equivalence relations (i.e. kernel pairs) Q and P on X, maintaining the invariant that P is finer than Q. In each step, we take into account new information on the behaviour of states, represented by a map $q : X \to K$, and accumulate this information in a map $\bar{q} : X \to \bar{K}$. In order to facilitate our analysis further below, these variables are indexed over loop iterations in the description. Initial values are

$$Q_0 = X \times X \qquad q_0 = ! : X \to 1 = K_0 \qquad P_0 = \mathsf{ker}(X \xrightarrow{\xi} HX \xrightarrow{H!} H1).$$

We then iterate the following steps while $P_i \neq Q_i$, for $i \geq 0$:

(1) $X/P_i \overset{k_{i+1}}{\twoheadrightarrow} K_{i+1} := \mathsf{select}\big(X \overset{\kappa_{P_i}}{\twoheadrightarrow} X/P_i \twoheadrightarrow X/Q_i\big)$, using that X/P_i is finer than X/Q_i

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

(3) $Q_{i+1} := \ker \bar{q}_{i+1}$ $(= \ker\langle \bar{q}_i, q_{i+1}\rangle = \ker \bar{q}_i \cap \ker q_{i+1},$ see (2.1))

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

Upon termination, the algorithm returns $X/P_i = X/Q_i$ as the simple quotient of (X, ξ).

Note that this is precisely the informal Algorithm 3.1 where:

- The partitions and equivalence relations are replaced by coequalizers and kernel pairs.
- Transition type specific steps involve how the type functor H acts on morphisms.
- The choice of subblock S and compound block C is replaced by the select routine.

We proceed to prove correctness, i.e. that the algorithm computes simple quotients of coalgebras. We use the notation in Algorithm 4.5 throughout. Since $\bar{q}$ accumulates more information in every step, it is clear that P and Q are being successively refined:

Lemma 4.6. *For every i, P_{i+1} is finer than P_i, Q_{i+1} is finer than Q_i, and P_i is finer than Q_{i+1}.*

$$\begin{array}{ccccccccccc} Q_0 & \longleftarrow & Q_1 & \longleftarrow & Q_2 & \leftarrow\cdots\prec & Q_{i+1} & \longleftarrow & Q_{i+2} & \leftarrow\cdots \\ \updownarrow & & \updownarrow & & \updownarrow & & \updownarrow & & \updownarrow & \\ P_0 & \longleftarrow & P_1 & \leftarrow\cdots\prec & P_i & \longleftarrow & P_{i+1} & \leftarrow\cdots & & \end{array} \tag{4.1}$$

Proof. We use Remark 2.4(1) to show that one kernel is finer than the other:

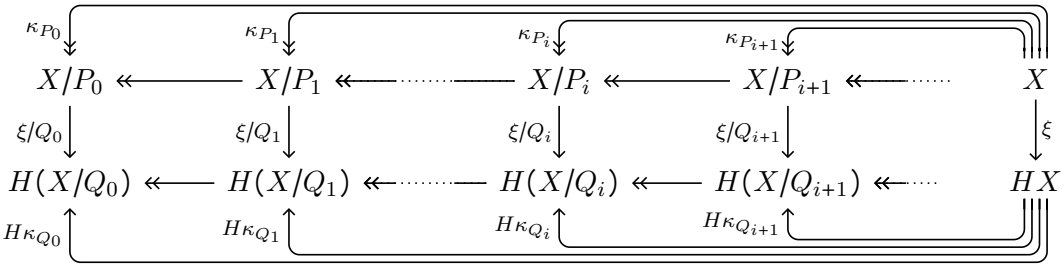
(1) P_{i+1} finer than P_i and Q_{i+1} finer than Q_i: Let $p : \prod_{j \le i+1} K_j \to \prod_{j \le i} K_j$ be the product projection. Clearly we have $\bar{q}_i = p \cdot \bar{q}_{i+1}$, so $Q_{i+1} = \ker(\bar{q}_{i+1})$ is finer than $Q_i = \ker \bar{q}_i$. Similarly $H\bar{q}_i \cdot \xi = Hp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

(2) P_i finer than Q_{i+1}: Induction on i. Since $Q_{i+1} = \ker \bar{q}_{i+1}$ and $\bar{q}_{i+1} = \langle q_0, \ldots, q_{i+1}\rangle$, it suffices to show that P_i is finer than $\ker q_j$ for $j = 0, \ldots, i+1$. For $j \le i$, we have by item (1) that P_i is finer than P_j, which is finer than $\ker q_j$ by induction. Moreover, P_i is finer than $\ker q_{i+1}$ because q_{i+1} factors through $X \twoheadrightarrow X/P_i$ by construction. □

Ignoring the termination when $P_i = Q_i$ for a moment, the algorithm computes equivalence relations refining each other. In each step, select decides which part of the information present in P_i but not in Q_i should be used to refine Q_i to Q_{i+1}.

Proposition 4.7. *There exist morphisms $\xi/Q_i : X/P_i \to H(X/Q_i)$ for $i \ge 0$ (necessarily unique) such that* (4.2) *commutes.*

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



Proof. Since $Q_i = \ker \bar{q}_i$, the image factorization of $\bar{q}_i$ has the form $\bar{q}_i = m \cdot \kappa_{Q_i}$. By definition of P_i and because H preserves monos, we thus have $P_i = \ker(H\bar{q}_i \cdot \xi) = \ker(H\kappa_{Q_i} \cdot \xi)$, and hence obtain ξ/Q_i as in (4.2) by the universal property of the coequalizer κ_{P_i}. □

Upon termination the morphism ξ/Q_i yields the structure of a quotient coalgebra of ξ:

Corollary 4.8. *If $P_i = Q_i$ then X/Q_i carries a unique coalgebra structure forming a quotient of $\xi : X \to HX$.*

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:

Theorem 4.9 (Correctness)**.** *If $P_i = Q_i$, then $\xi/Q_i \colon X/Q_i \to H\,X/Q_i$ is a simple coalgebra.*

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

(1) We first prove that

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

This is seen as follows: If $\ker h$ is finer than Q_i, then $\kappa_{Q_i} \colon X \to X/Q_i$ factors through $h \colon X \twoheadrightarrow D$, so that $H\kappa_{Q_i} \cdot \xi$ factors through $Hh \cdot \xi$ and hence through h, since $Hh \cdot \xi = d \cdot h$:

$$\begin{array}{ccccc} X & \xrightarrow{\xi} & HX & \xrightarrow{H\kappa_{Q_i}} & H(X/Q_i) \\ \downarrow{\scriptstyle h} & & \downarrow{\scriptstyle Hh} & \nearrow_{Hq} & \\ D & \xrightarrow{d} & HD & & \end{array}$$

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

(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). By induction hypothesis $\ker(h)$ is finer than P_i, thus by Lemma 4.6 also finer than Q_{i+1} and consequently by (4.3) also 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 h is an isomorphism, since kernels and their respective quotients are in bijective correspondence. □

Remark 4.10. Most classical partition refinement algorithms are parametrized by an initial partition $\kappa_{\mathcal{I}} : X \twoheadrightarrow X/\mathcal{I}$. We start with the trivial partition $! : X \to 1$ because a non-trivial initial partition might split equivalent behaviours and then would invalidate Theorem 4.9. To accomodate 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 4.11. (1) Recall that H induces the *final sequence*:

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

Every coalgebra $\xi : X \to HX$ then induces a *canonical cone* $\xi^{(i)} : X \to H^i 1$ on the final sequence, defined inductively by $\xi^{(0)} = !$, $\xi^{(i+1)} = H\xi^{(i)} \cdot \xi$. 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 *finitary* set functors, i.e. set functors preserving filtered colimits; equivalently, a set functor H is finitary if for every $x \in HX$ there exists a finite subset $m : Y \rightarrowtail X$ and $y \in HY$ such that $x = Hm(y)$. Note that for *finite* coalgebras 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

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

Indeed, 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.

(2) The vertical inclusions $P_i \rightarrowtail Q_{i+1}$ in (4.1) 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, i.e., we have $k_{i+1} := \mathsf{id}_{X/P_i}$, then these inclusions become isomorphisms and then our algorithm simply computes the kernels of the morphisms in the canonical cone, i.e. $Q_i = \ker \xi^{(i)}$: we have $q_{i+1} = \kappa_{P_i} : X \to X/P_i$ for all i, so the q_i successively refine each other, so that $Q_{i+1} = \ker \bar{q}_{i+1} = \ker\langle q_0, \ldots, q_{i+1}\rangle = \ker q_{i+1} = P_i$.

We show $Q_i = \ker \xi^{(i)}$ for $i \geq 0$ by induction on i, with trivial base case. For the inductive step, note that from $\ker \bar{q}_i = \ker q_i$ and because q_i is a regular epi we obtain a mono m such that $\bar{q}_{i+1} = m \cdot q_{i+1}$ by Remark 2.4(3); similarly, the induction hypothesis implies that we have a mono n such that $\xi^{(i)} = n \cdot q_i$. Since H preserves monomorphisms, this implies that

$$\begin{aligned} Q_{i+1} = P_i &= \ker(H\bar{q}_i \cdot \xi) = \ker(Hm \cdot Hq_i \cdot \xi) = \ker(Hq_i \cdot \xi) \\ &= \ker(Hn \cdot Hq_i \cdot \xi) = \ker(H\xi^{(i)} \cdot \xi) = \ker(\xi^{(i+1)}). \end{aligned}$$

That is, when select retains all available information, then Algorithm 4.5 just becomes a standard final chain algorithm (e.g. [KK14]). The other extreme is the following:

Definition 4.12. We say that select *discards all new information at* y, z if $k = \mathsf{select}(X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z)$ factors through z. Further, we call select *progressing* if z is an isomorphism whenever select discards all information at y, z.

Example 4.13. (1) The select picking the smaller half in Example 4.4(1) is progressing. We prove the contraposition: if the z in $X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z$ is not an isomorphism, then a subblock $S \in Y$ is found and by the size constraint there is a distinct block $B \in Y$ with $z(B) = z(Y)$. 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 $X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z$ (Example 4.4(2)) is trivially progressing: if id_X factors through z, then z is a split-mono, and thus an isomorphism.

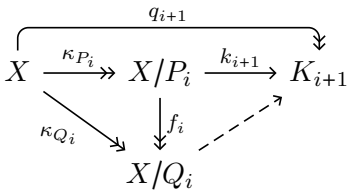
(3) The select routine that returns the terminal morphism $!: Y \to 1$ or $z: Y \to Z$ always discards all new information, and thus is not progressing (in non-singleton categories).

Theorem 4.14. *If* select *is progressing, then Algorithm 4.5 terminates and computes the simple quotient of a given coalgebra provided it 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 $i+1$-st iteration, i.e. $Q_{i+1} = Q_i$, iff select discards all new information at $X/P_i, X/Q_i$.

In order to see this, first note that select discards all new information at $X/P_i, X/Q_i$ iff $q_{i+1} = k_{i+1} \cdot \kappa_{P_i}$ factors through $f_i \cdot \kappa_{P_i} = \kappa_{Q_i}$.



Since Q_{i+1} is finer than Q_i by Lemma 4.6, it suffices to reason as follows: $Q_i = \ker \bar{q}_i$ is finer than $Q_{i+1} = \ker\langle \bar{q}_i, q_{i+1}\rangle$ iff $Q_i = \ker(\kappa_{Q_i})$ is finer than $\ker q_{i+1}$ iff q_{i+1} factors through κ_{Q_i} (for the last step see Remark 2.4(4)).

(2) We proceed to prove the claim. Lemma 4.6 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 discards all new information at $X/P_i, X/Q_i$. The select routine is progressing, so we obtain $P_i = Q_i$ as desired. □

5. Incremental Partition Refinement

In the most generic version of the partition refinement algorithm (Algorithm 4.5), the partitions are recomputed from scratch in every step: In Step 4 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). The algorithm will see a further refinement in the next section.

Note that in Step 3, Algorithm 4.5 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. (2.1)). 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 $q_{i+1} : X \to K_{i+1}$.

Algorithm 4.5 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 the following, we will provide sufficient conditions for H, $a : D \to A$, $b : D \to B$ to satisfy

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

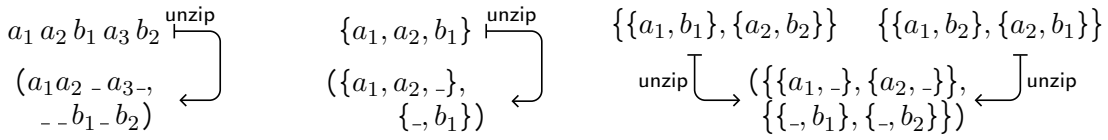
As soon as this holds for $a = \bar{q}_i, b = q_{i+1}$, we can optimize the algorithm by changing Step 4 to

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

Definition 5.1. A functor H is *zippable* if the following morphism is a monomorphism:

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

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 in the examples in Figure 1.



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

FIGURE 1. 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 $f : A \to C$, $g : B \to D$. 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

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

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

In the following, we work in $\mathcal{C} = \mathsf{Set}$. However, most proofs are category-theoretic to clarify where working in Set is really needed and where the arguments are more general.

Example 5.3. (1) Constant functors $X \mapsto A$ are zippable: unzip is the diagonal $A \to A \times A$.
(2) The identity functor is zippable since $\langle A+!, !+B\rangle : A+B \to (A+1) \times (1+B)$ is monic in Set.
(3) From Lemma 5.4 it follows that every polynomial endofunctor is zippable.

Lemma 5.4. *Zippable endofunctors are closed under products, coproducts and subfunctors.*

(Recall that products and coproducts of functors are formed pointwise, e.g. $(H_1 + H_2)(X) = H_1X + H_2X$.)

Proof. 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:

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

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

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

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 $s : F \rightarrowtail G$, where G is zippable. Then the following diagram shows that F is zippable, too:

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

□

Lemma 5.5. *If H has a componentwise monic natural transformation $H(X + Y) \rightarrowtail HX \times HY$, then H is zippable.*

Proof. Let $\alpha_{X,Y} : H(X + Y) \rightarrowtail HX \times HY$ 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 5.6. (1) For every commutative monoid, the monoid-valued functor $M^{(-)}$ admits a natural isomorphism $M^{(X+Y)} \cong M^{(X)} \times M^{(Y)}$, and hence is zippable by Lemma 5.5.

(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) 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 5.4.

(4) The previous examples together with the closure properties in Lemma 5.4 show that a number of functors of interest are zippable, e.g. $2 \times (-)^A$, $2 \times \mathcal{P}(-)^A$, $\mathcal{P}(A \times (-))$, $2 \times \big((-) + 1\big)^A$, and variants where $\mathcal{P}$ is replaced by $\mathcal{B}_\mathrm{f}$, $M^{(-)}$, or $\mathcal{D}$.

Remark 5.7. Out of the above results, only zippability of the identity and coproducts of zippable functors depend on our assumptions on $\mathcal{C}$ (see Assumption 2.2). Indeed, zippable functors are closed under coproducts as soon as monomorphisms are closed under coproducts,

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]). 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 7).

Example 5.8. The monotone neighbourhood functor mapping a set X to

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

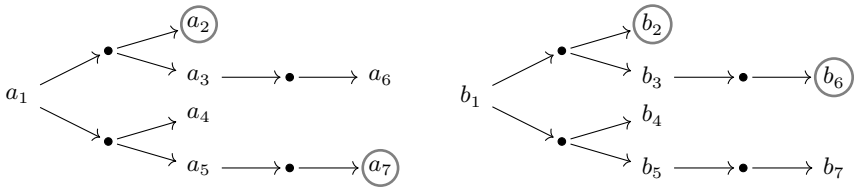
is not zippable. There are neighbourhoods which are identified by unzip; indeed let $A = \{a_1, a_2\}$, $B = \{b_1, b_2\}$ and denote by $(-)\uparrow$ the upwards-closure:

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

Example 5.9. The functor $\mathcal{P}_\mathrm{f}\mathcal{P}_\mathrm{f}$ fails to be zippable, as shown in Figure 1. First, this shows that zippable functors are not closed under quotients, since $\mathcal{P}_\mathrm{f}$ is a quotient of the polynomial, hence zippable, functor $HX = \coprod_{n<\omega} X^n$. Secondly, this shows that zippable functors are not closed under composition.

The following example shows that the optimized algorithm 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 4.4(1)) behaves sufficiently well (specified later in Definition 5.12 and cf. Corollary 5.18).

Example 5.10. Consider the following coalgebra $\xi : X \to HX$ for $HX = 2 \times \mathcal{P}_\mathrm{f}\mathcal{P}_\mathrm{f}X$:



States x with $\pi_1(\xi(x)) = 1$ are indicated by a circle. When computing only

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

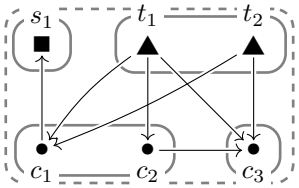
instead of $\ker(H\langle q_{i+1}, \bar{q}_i\rangle \cdot \xi)$, then the states a_1 and b_1 are not distinguished, although they are behaviourally inequivalent.

If we simplify the partitions by defining abbreviations for the circle and non-circle states without successors as well as the rest,

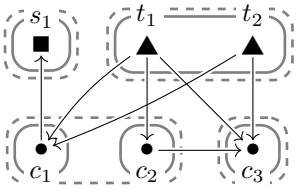
$$F := \{a_2, a_7, b_2, b_6\}, \quad N := \{a_4, a_6, b_4, b_7\} \quad \text{and} \quad C := \{a_1, a_3, a_5, b_1, b_3, b_5\},$$

running the optimized algorithm, i.e. computing Q_i and P'_i, the result is the following sequence of partitions.

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

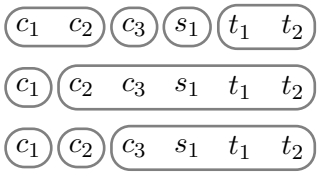


(A) Q_0, P_0 for $\bar{q}_0 = !$



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

FIGURE 2. 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.



$X/Q_1 = X/P_0$

$X/\ker \chi_S$

$X/\ker \chi_S^C$

FIGURE 3. Grouping of elements when $S := \{c_1\}$ is chosen as the next subblock and $C := \{c_1, c_2\}$ as the compound block.

Note that in step $i = 3$ the result is the same for $S' := \{a_5, b_3\}$, and also if we split by $\{a_3, b_5\}$ in step 2 and by $\{a_1, b_1\}$ in step 3. For $S = \{a_3, b_5\}$ as in the above table, $\{a_1, b_1\}$ is not split in X/P_3' because:

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

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

Observe that, in general, $\ker H\langle a, b\rangle$ differs from $\ker\langle Ha, Hb\rangle$ even if H is zippable; e.g. for $H = \mathcal{P}$ and for π_1, π_2 denoting binary product projections, $\langle \mathcal{P}\pi_1, \mathcal{P}\pi_2\rangle$ in general fails to be injective although $\mathcal{P}\langle \pi_1, \pi_2\rangle = \mathcal{P}\mathsf{id} = \mathsf{id}$.

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 $q_i = \chi_S$. 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 5.11. Consider the coalgebra $\xi\colon X \to HX$ for the zippable functor $H = \{\blacktriangle, \blacksquare, \bullet\} \times \mathcal{P}_\mathrm{f}(-)$ illustrated in Figure 2 (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 2a). Now, suppose that select returns $k_1 := \mathsf{id}_{X/P_0}$, i.e.

retains all information (cf. Remark 4.11), so that $Q_1 = P_0$ and P_1 puts c_1 and c_2 into different blocks (Figure 2b). 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 $k_2 := \chi_{\{\{c_1\}\}} : X/P_1 \to 2$, making $q_2 = \chi_{\{c_1\}} : X \to 2$. 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 $X/\ker \chi_S$. Formally:

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

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 4.4(1)). This is formally captured by the condition we introduce next.

Definition 5.12. 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. We can rephrase the condition more explicitly:

Lemma 5.13. *For $a : D \to A$, $b : D \to B$ in Set, the following are equivalent:*

(1) $\ker a \cup \ker b \rightrightarrows D$ *is a kernel (i.e. an equivalence relation).*
(2) $\ker a \cup \ker b \rightrightarrows D$ *is the kernel of the pushout of a and b.*
(3) *For all $x, y, z \in D$, $a(x) = a(y)$ and $b(y) = b(z)$ implies $a(x) = a(y) = a(z)$ or $b(x) = b(y) = b(z)$.*
(4) *For all $x \in D$, $[x]_a \subseteq [x]_b$ or $[x]_b \subseteq [x]_a$.*

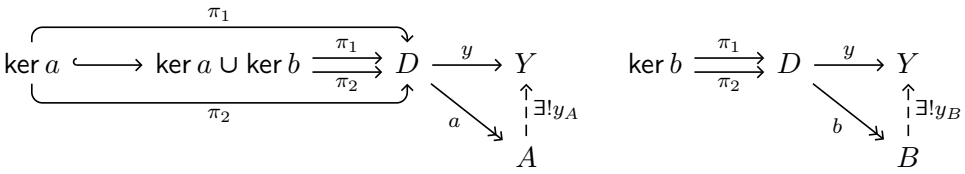
The last item states that when we move from a-equivalence classes to b-equivalence classes, the classes either merge or split, but do not merge with other classes and split at the same time. Note that in Figure 3, $Q_1 \cup \ker \chi_S$ fails to be transitive, while $Q_1 \cup \ker \chi_S^C$ is transitive.

Proof. (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:

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

Let $\ker a \cup \ker b$ be the kernel of some $y\colon D \to Y$. Then, y makes the projections of $\ker a$ (resp. $\ker b$) equal and hence the coequalizer a (resp. b) induces a unique y_A (resp. y_B):



Because of $y_B \cdot b = y = y_A \cdot a$, (y_A, y_B) is a competing cocone for the above pushout. This induces a cocone morphism $y_P : (P, p_A, p_B) \to (Y, y_A, y_B)$, and we have

$$y_P \cdot p = y_P \cdot p_A \cdot a = y_A \cdot a = y.$$

With this, we are ready to show that $\ker a \cup \ker b$ is a kernel for p. Consider two morphisms $c_1 : C \to D$, $c_2 : C \to D$ with $p \cdot c_1 = p \cdot c_2$, then we have

$$y \cdot c_1 = y_P \cdot p \cdot c_1 = y_P \cdot p \cdot c_2 = y \cdot c_2.$$

This induces a unique cone morphism $C \to \ker a \cup \ker b$ as desired.

(2) $\Rightarrow$ (3) Take $x, y, z \in D$ 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 $y \in D$, 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 $x \in [y]_a$, i.e. such that $a(x) = a(y)$, with $b(x) \neq b(y)$. 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. $z \in [y]_a$. □

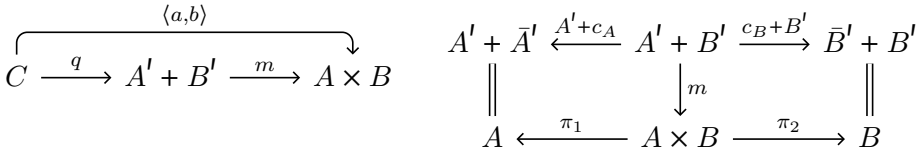
Example 5.14. All select routines in Example 4.4 respect compound blocks. To see this, let $k = \mathsf{select}\big(X \overset{y}{\twoheadrightarrow} Y \overset{z}{\twoheadrightarrow} Z\big)$.

(1) For $S \in Y$ and $C := [S]_z$, $k := \chi^C_{\{S\}}$ respects compound blocks by Lemma 5.13(4):
   - For $p \in Y \setminus C$, $z(p) \neq z(S)$ and so $[p]_z \subseteq Y \setminus C = [p]_k$.
   - For $p \in C$, $z(p) = z(S)$ and so $[p]_k \subseteq C = [p]_z$.

(2) The select routine returning the identity respects compound blocks, because for any morphism $a : D \to A$, $\ker \mathsf{id}_D \cup \ker a = \ker a$ is a kernel.

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

Lemma 5.15. *Let $a\colon C \to A$, $b\colon C \to B$ such that $\ker a \cup \ker b$ is a kernel, then there exist appropriate sets and maps making the following diagrams commute:*



Proof. Define the sets

$$C_A = \{x \in C \mid [x]_a \subseteq [x]_b\}, \qquad C_B = \{x \in C \mid [x]_a \supsetneq [x]_b\},$$
$$A' = \{[x]_a \mid x \in C_A\}, \qquad B' = \{[x]_b \mid x \in C_B\}.$$

By Lemma 5.13, $C = C_A + C_B$. Next define $q = a' + b' : C \cong C_A + C_B \to A' + B'$, where a' and b' are the obvious restrictions of a and b, respectively. Now define $\bar{A}' := A \setminus A'$, $\bar{B}' := B \setminus B'$ and

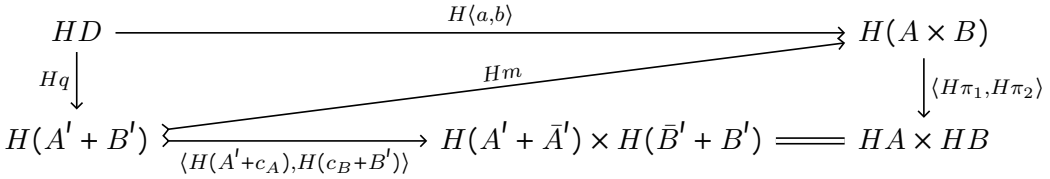
$$c_A([x]_b) = a(x) \quad \text{and} \quad c_B([x]_a) = b(x).$$

The functions are well-defined by definition of C_A and C_B. The codomain of c_A restricts to $\bar{A}'$, because $a(x) \in A'$ implies that $[x]_a \subseteq [x]_b$, contradicting $x \in C_B$ – and the codomain c_B restricts to $\bar{B}'$, analogously. Hence, $(\mathsf{id}_{A'} + c_A) \cdot q = a$ and $(c_B + \mathsf{id}_{B'}) \cdot q = b$, and so m is induced by the product $A \times B$ making both desired diagrams commute. □

Proposition 5.16. *Let $a\colon D \to A$, $b\colon D \to B$ such that $\ker a \cup \ker b$ is a kernel, and let $H : \mathsf{Set} \to \mathcal{D}$ be a zippable functor. Then we have*

$$\ker\langle Ha, Hb\rangle = \ker H\langle a, b\rangle. \tag{5.2}$$

Proof. Using the additional data as provided by Lemma 5.15, we note that the following diagram commutes:



Note that the composition on the bottom is a mono, because H is zippable (cf. Lemma 5.2). Hence, Hm is a mono as well. We can conclude using Remark 2.4(2) for the last two equations below:

$$\ker\langle Ha, Hb\rangle = \ker\big(\langle H\pi_1, H\pi_2\rangle \cdot H\langle a, b\rangle\big) = \ker(Hq) = \ker H\langle a, b\rangle. \qquad \square$$

Theorem 5.17. *If $H : \mathsf{Set} \to \mathsf{Set}$ is zippable and* select *respects compound blocks, then optimization* (5.1) *is correct.*

Proof. Correctness of (5.1) means that

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

Indeed, for H zippable, k_i as in Algorithm 4.5 and $f_i : X/P_i \twoheadrightarrow X/Q_i$ witnessing that P_i is finer than Q_i, we have:

$$\begin{aligned}
& \text{select respects compound blocks} \\
\overset{\text{Def.}}{\iff}\ & \ker f_i \cup \ker k_{i+1} \text{ is a kernel} \\
\overset{5.16}{\implies}\ & \ker\langle Hf_i, Hk_{i+1}\rangle = \ker H\langle f_i, k_{i+1}\rangle \\
\overset{2.4(5)}{\implies}\ & \ker(\langle Hf_i, Hk_{i+1}\rangle \cdot H\kappa_{P_i} \cdot \xi) = \ker(H\langle f_i, k_{i+1}\rangle \cdot H\kappa_{P_i} \cdot \xi) \\
\implies\ & P_i \cap \ker(Hq_{i+1} \cdot \xi) = \ker\big(\langle H\bar{q}_i \cdot \xi, Hq_{i+1} \cdot \xi\rangle\big) = \ker\big(H\langle \bar{q}_i, q_{i+1}\rangle \cdot \xi\big) \\
& \qquad\qquad = \ker\big(H\bar{q}_{i+1} \cdot \xi\big) = P_{i+1}
\end{aligned}$$

□

Theorem 4.14 now yields:

Corollary 5.18. *Suppose that H is a zippable endofunctor and that* select *respects compound blocks and is progressing. Then Algorithm 4.5 with optimization* (5.1) *terminates and computes the simple quotient of a given finite H-coalgebra.*

Remark 5.19. 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 5.13 and Lemma 5.15 can be formulated in the internal language of a Boolean topos, i.e. the ordinary set theory ZF (with guarded quantifiers only) without the axiom of choice, but with the law of excluded middle. An detailed definition and discussion of this language can be found in e.g. Mac Lane and Moerdijk [LM92].

6. Efficient Calculation of Kernels

In Algorithm 4.5, we left many details unspecified, especially how to compute kernels. We now proceed to define a more detailed algorithm parametric in a *refinement interface* for the given type functor. This interface is aimed towards an efficient implementation of the *refinement step* in the algorithm. Specifically, from now on, in lieu of a coalgebra we split along a map $\xi\colon X \to HY$ w.r.t. a subblock $S \subseteq C \in Y/Q$, and need to compute the changes to the partition X/P arising from splitting the compound block C into S and $C \setminus S$ within Y/Q. Throughout this section we work in Set and, in order to have such a subblock situation $S \subseteq C$ and to obtain a low complexity, we fix the select routine given in Example 4.4(1).

The low complexity of Paige-Tarjan-style algorithms hinges on the refinement step running in time $\mathcal{O}(|\mathsf{pred}[S]|)$, where $\mathsf{pred}(y)$ denotes the set of predecessors of a $y \in Y$ in the given transition system. In order to speak about "predecessors" w.r.t. a map $\xi : X \to HY$, a refinement interface for H will provide an encoding of ξ as a set of states with successor structures encoded as bags (implemented as lists up to ordering; recall from Example 2.6(3) that $\mathcal{B}_\mathrm{f}Z$ denotes the set of bags over Z) of A-labelled edges, where A is an appropriate label alphabet. Moreover, this interface will allow us to analyse the behaviour of elements of X w.r.t. the splitting of C into S and $C \setminus S$ simply by looking at elements in S.

Definition 6.1. A *refinement interface* for a Set-functor H is formed by a set A of *labels*, a set W of *weights* and functions

$$\begin{aligned} &\mathsf{b} : HY \to \mathcal{B}_\mathrm{f}(A \times Y), && \mathsf{init} : H1 \times \mathcal{B}_\mathrm{f}A \to W,\\ &w : \mathcal{P}_\mathrm{f}Y \to HY \to W, && \mathsf{update} : \mathcal{B}_\mathrm{f}A \times W \to W \times H3 \times W \end{aligned}$$

such that for every $S \subseteq C \subseteq Y$, the diagrams

$$\begin{array}{ccc} HY & \xrightarrow{w(Y)} & \\ {\scriptstyle\langle H!,\, \mathcal{B}_\mathrm{f}\pi_1 \cdot \mathsf{b}\rangle}\downarrow & & \\ H1 \times \mathcal{B}_\mathrm{f}A & \xrightarrow{\mathsf{init}} & W \end{array} \qquad \begin{array}{ccccc} HY & & \xrightarrow{\langle w(S), H\chi_S^C, w(C\setminus S)\rangle} & & \\ {\scriptstyle\langle \mathsf{b}, w(C)\rangle}\downarrow & & & & \downarrow \\ \mathcal{B}_\mathrm{f}(A\times Y)\times W & \xrightarrow{\mathsf{fil}_S \times W} & \mathcal{B}_\mathrm{f}A \times W & \xrightarrow{\mathsf{update}} & W \times H3 \times W \end{array} \tag{6.1}$$

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

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 $s \in B$ by the value of $H\chi_S^C \cdot \xi(s) \in H3$. The set W depends on the functor. But in most cases $W = H2$ and $w(C) = H\chi_C : HY \to H2$ are sufficient.

In an implementation, we do not require a refinement interface to provide w explicitly, because the algorithm will compute the values of w incrementally using (6.1), and b need not be implemented because we assume the input coalgebra to be already *encoded* via b:

Definition 6.2. Given an interface of H (Definition 6.1), an *encoding* of a morphism $\xi : X \to HY$ is given by a set E and maps

$$\mathsf{graph} : E \to X \times A \times Y \qquad\qquad \mathsf{type} : X \to H1$$

such that $(\flat \cdot \xi(x))(a, y) = |\{e \in E \mid \mathsf{graph}(e) = (x, a, y)\}|$, and with $\mathsf{type} = H! \cdot \xi$.

Intuitively, an encoding presents the morphism ξ as a graph with edge labels from A.

Lemma 6.3. *Every morphism $\xi : X \to HY$ has a canonical encoding where E is the obvious set of edges of $\flat \cdot \xi : X \to \mathcal{B}_{\mathrm{f}}(A \times Y)$. If X is finite, then so is E.*

Proof. Define E as follows. Compose ξ with $\flat$ and the inclusion into the set of all maps $A \times Y \to \mathbb{N}$:

$$X \xrightarrow{\xi} HY \xrightarrow{\flat} \mathcal{B}_{\mathrm{f}}(A \times Y) = \mathbb{N}^{(A \times Y)} \hookrightarrow \mathbb{N}^{A \times Y}.$$

Its uncurrying is a map $\mathsf{cnt} : X \times A \times Y \to \mathbb{N}$, and we let

$$E := \coprod_{e \in X \times A \times Y} \mathsf{cnt}(e),$$

where each $\mathsf{cnt}(e) \in \mathbb{N}$ is considered as a finite ordinal number. By copairing we then obtain a unique morphism $\mathsf{graph} : E \to X \times A \times Y$ defined on the coproduct components as

$$\mathsf{cnt}(e) \xrightarrow{!} 1 \xrightarrow{e} X \times A \times Y,$$

and we put $\mathsf{type} = H! \cdot \xi$. Note that if X is finite, then so is E, since all $\flat \cdot \xi(x)$ are finitely supported. □

Example 6.4. In the following examples, we take $W = H2$ and $w(C) = H\chi_C : HY \to H2$. We use the helper function $\mathsf{val} := \langle H(= 2), \mathsf{id}, H(= 1)\rangle : H3 \to H2 \times H3 \times H2$, where $(= x) : 3 \to 2$ is the equality check for $x \in \{1, 2\}$, and in each case define $\mathsf{update} = \mathsf{val} \cdot \mathsf{up}$ for the function $\mathsf{up} : \mathcal{B}_{\mathrm{f}}A \times H2 \to H3$ is defined individually for every functor. We implicitly convert sets into bags.

For the verification of (6.1) note that, in general, for $S \subseteq C \subseteq Y$, 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 $\mathsf{up} \cdot \langle \mathsf{fil}_S \cdot \flat, H\chi_C\rangle = H\chi^C_S$; in fact, using $w(C) = H\chi_C$ we have:

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

(1) For the monoid-valued functor $H = G^{(-)}$, for an Abelian group $(G, +, 0)$, we take labels $A = G$ and define $\flat(f) = \{(f(y), y) \mid y \in Y, f(y) \neq 0\}$ (which is is finite because f is finitely supported). With $W = H2 = G \times G$, the weight $w(C) = H\chi_C : HY \to G \times G$ assigns to $f \in HY = G^{(Y)}$ the pair of accumulated weights of $Y \setminus C$ and C under f. Then 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 $\Sigma : \mathcal{B}_{\mathrm{f}}G \to G$ is the obvious summation map.

Indeed, for any $f \in HY = G^{(Y)}$, we have:

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

(2) Similarly, from the interface for $\mathbb{R}^{(-)}$ we can derive for the to the distribution functor $\mathcal{D}$, a subfunctor of $\mathbb{R}_{\geq 0}^{(-)}$, the following init and up functions:

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

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 $f \in \mathcal{D}Y$, we have $\Sigma\mathcal{B}\pi_1 \cdot \mathsf{b}(f) = \sum_{y\in Y} f(y) = 1$.

The axiom for up is proved as in the previous example; in fact, note that for an $f \in \mathcal{D}Y$ all components of the triple $\big(\sum_{y\in Y\setminus C} f(y), \sum_{y\in C\setminus S} f(y), \sum_{y\in S} f(y)\big)$ are in $[0,1]$ and their sum is $\sum_{y\in Y} f(y) = 1$. 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 example (2) above: if f lies in $\mathbb{N}^{(Y)}$ then $\sum_{y\in Y} f(y)$ lies in $\mathbb{N}$ and so do the components of the triple in the proof of the axiom for up, whence we obtain $\mathbb{N}^{(\chi_S^C)}(f)$.

(4) Given a polynomial functor H_Σ for the signature Σ with bounded arity (i.e. there exists an upper bound on the arity of operation symbols in Σ), the labels $A = \mathbb{N}$ encode the indices of the parameters:

$$\mathsf{b}(\sigma(y_1,\dots,y_n)) = \{(1,y_1),\dots,(n,y_n)\} \quad \mathsf{init}(\sigma(0,\dots,0), f) = \sigma(1,\dots,1)$$
$$\mathsf{up}(I, \sigma(b_1,\dots,b_n)) = \sigma(b_1 + (1 \in I), \dots, b_i + (i \in I), \dots, b_n + (n \in I))$$

Here $b_i + (i \in I)$ means $b_i + 1$ if $i \in I$ and b_i otherwise. Since I is the set of indices of the parameters in the subblock, $i \in I$ happens only if $b_i = 1$.

Let $t = \sigma(y_1, \dots, y_n) \in H_\Sigma Y$ with σ of arity n, let $c_i = \chi_C(y_i)$, and $I = \{1 \le i \le n \mid y_i \in S\}$. Then we have:

$$\begin{aligned}
\mathsf{init}(H_\Sigma !(t), \mathcal{B}_\mathrm{f}\pi_1 \cdot \mathfrak{b}(t)) &= \mathsf{init}(\sigma(0,\dots,0), \mathcal{B}_\mathrm{f}\pi_1(\{(1,y_1),\dots,(n,y_n)\})) \\
&= \mathsf{init}(\sigma(0,\dots,0), \{1,\dots,n\}) = \sigma(1,\dots,1) \\
&= \sigma(\chi_Y(y_1),\dots,\chi_Y(y_n)) = H_\Sigma\chi_Y(t). \\
\mathsf{up}(\mathsf{fil}_S \cdot \mathfrak{b}(t), H_\Sigma\chi_C(t)) &= \mathsf{up}(\mathsf{fil}_S(\{(1,y_1),\dots,(n,y_n)\}), \sigma(c_1,\dots,c_n)) \\
&= \mathsf{up}(I, \sigma(c_1,\dots,c_n)) \\
&= \sigma(c_1 + (1 \in I), \dots, c_i + (i \in I), \dots, c_n + (n \in I)) \\
&= \sigma(\chi_S^C(y_1), \dots, \chi_S^C(y_i), \dots, \chi_S^C(y_n)) \\
&= H_\Sigma\chi_S^C(t).
\end{aligned}$$

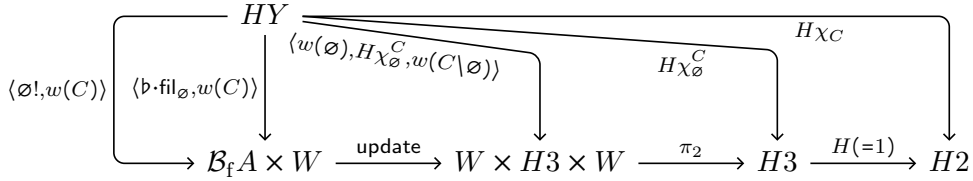
In the penultimate step we use that:

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

The functor-specific $w(C)$ can not always be $H\chi_C$, but there is the following connection:

Proposition 6.5. *For any 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:



□

One example where $W = H2$, and thus $w(C) = H\chi_C$, does not suffice is the powerset functor $\mathcal{P}$: Even if we know for a $t \in \mathcal{P}Y$ that it contains elements in $C \subseteq Y$, in $S \subseteq C$, and outside C (i.e. we know $\mathcal{P}\chi_S(t), \mathcal{P}\chi_C \in \mathcal{P}2$), we cannot determine whether there are any elements in $C \setminus S$ – but as seen in Example 5.11, we need to include this information.

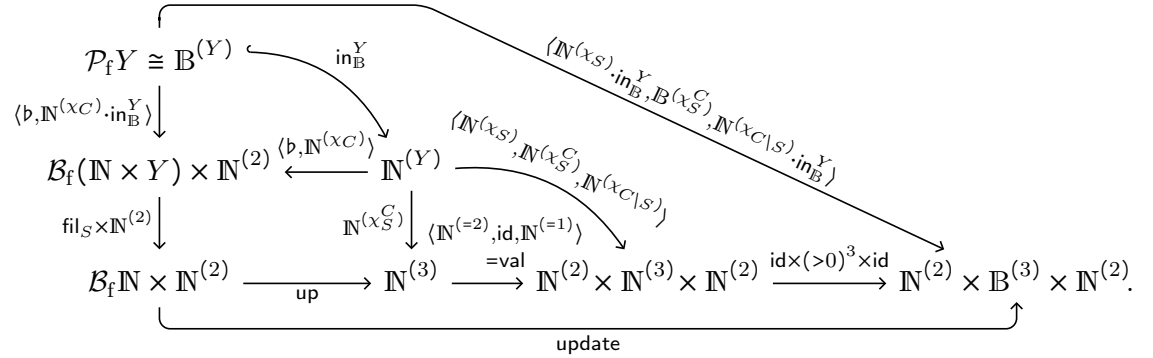
Example 6.6. The refinement interface for the powerset functor needs to count the edges into blocks $C \supseteq S$ in order to know whether there are edges into $C \setminus S$, as described by Paige and Tarjan [PT87]. What happens formally is that the interface for $\mathbb{N}^{(-)}$ is implemented for edge weights of at most 1, and then the middle component of the result of update is adjusted. So $W = \mathbb{N}^{(2)}$, $A = \mathbb{N}$ (but only the weight 1 will appear), and the encoding $\mathfrak{b} : \mathcal{P}_\mathrm{f}Y \hookrightarrow \mathcal{B}_\mathrm{f}(1 \times Y) \hookrightarrow \mathcal{B}_\mathrm{f}(\mathbb{N} \times Y)$ is the obvious inclusion. Then

$$\begin{aligned}
\mathsf{init}(h_1, e) &= (0, |e|) \\
w(C)(M) &= \mathcal{B}_\mathrm{f}\chi_C(M) = (|M \setminus C|, |C \cap M|) \\
\mathsf{update}(n, (r,c)) &= \langle \mathcal{B}_\mathrm{f}(=2),\ (\overset{?}{>}0)^3,\ \mathcal{B}_\mathrm{f}(=1)\rangle(r, c-|n|, |n|) \\
&= \big((r + c - |n|, |n|), (r \overset{?}{>} 0, c - |n| \overset{?}{>} 0, |n| \overset{?}{>} 0), (r + |n|, c - |n|)\big),
\end{aligned}$$

where $x \overset{?}{>} 0$ is 0 if $x = 0$ and 1 otherwise.

To verify (6.1) we prove the axiom for init is analogously as for $\mathbb{N}^{(-)}$ in the proof for Example 6.4(1).

Note that we have $\mathsf{update} = \langle \mathbb{N}^{(=2)},\ (\overset{?}{>} 0)^3,\ \mathbb{N}^{(=1)} \rangle \cdot \mathsf{up}$, where $\mathsf{up} \colon \mathcal{B}_\mathrm{f}\mathbb{N} \times \mathbb{N}^{(2)} \to \mathbb{N}^{(3)}$ is as for $\mathbb{N}^{(-)}$. Now, we need to show the diagram below commutes:



The inner left-hand triangle clearly commutes. The square below it involving up and the middle lower triangle commute as shown in Example 6.4.1. The first and the third component of the remaining right-hand part clearly commute, and for the second component let $f \in \mathbb{B}^{(Y)}$ and compute as follows:

$$\begin{aligned}
\mathbb{B}^{\chi_S^C}(f) &= \left(\bigvee_{y \in Y \setminus C} f(y),\ \bigvee_{y \in C \setminus S} f(y),\ \bigvee_{y \in S} f(y) \right) \\
&= \left(0 \overset{?}{<} \sum_{y \in Y \setminus C} \mathsf{in}_\mathbb{B} \cdot f(y),\quad 0 \overset{?}{<} \sum_{y \in C \setminus S} \mathsf{in}_\mathbb{B} \cdot f(y),\quad 0 \overset{?}{<} \sum_{y \in S} \mathsf{in}_\mathbb{B} \cdot f(y) \right) \\
&= (\overset{?}{>} 0)^3 \cdot \left(g \mapsto \left(\sum_{y \in Y \setminus C} g(y),\ \sum_{y \in C \setminus S} g(y),\ \sum_{y \in S} g(y) \right) \right) \cdot \mathsf{in}_\mathbb{B}^Y(f) \\
&= (\overset{?}{>} 0)^3 \cdot \mathbb{N}^{(\chi_S^C)} \cdot \mathsf{in}_\mathbb{B}^Y(f).
\end{aligned}$$

Assumption 6.7. From now on, we assume that $H : \mathsf{Set} \to \mathsf{Set}$ is 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 6.8. We implictly make the usual assumptions 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 6.9. The refinement interfaces in Examples 6.4 and 6.6 satisfy Assumption 6.7.

Remark 6.10 (Details for Example 6.9)**.** For all those examples using the val-function from Example 6.4, 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.

For all the monoid-valued functors $G^{(-)}$ for an abelian group, for $\mathbb{N}$ and for $\mathcal{D}$, all the operations, including the summation Σe, run in time linear in the size of the input. If the elements $g \in G$ have a finite representation, so do the elements of $G^{(2)}$; thus comparing elements of $g_1, g_2 \in G^{(2)}$ can be performed in constant time.

For a polynomial functor H_Σ with bounded arities, we assume that operation symbols $\sigma \in \Sigma$ are encoded as integers; so we can assume that comparison of operation symbols runs in constant time. Since the signature has bounded arities, the maximum arity present in Σ is independent of any given morphism $X \to H_\Sigma Y$, so the comparison of the arguments of two flat Σ-terms $t_1, t_2 \in H_\Sigma 3$ runs in constant time as well.

(1) 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

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

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

(2) In $\mathsf{up}(I, \sigma(b_1, \dots, b_n))$, we cannot naively check all the $1 \in I, \dots, n \in I$ queries, since this would lead to a quadratic run-time. Instead we precompute all the queries' results together.

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

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

Remark 6.11. In the implementation, we encode the partitions X/P, Y/Q as doubly linked lists of the blocks they contain, and each block is in turn encoded as a doubly linked list of its elements. The elements $x \in X$ and $y \in Y$ each hold a pointer to the corresponding list entry in the blocks containing them. This allows removing elements from a block in $\mathcal{O}(1)$.

The algorithm maintains the following mutable data structures:

- An array $\mathsf{toSub} : X \to \mathcal{B}_\mathrm{f}E$, mapping $x \in X$ to its outgoing edges ending in the currently processed subblock.
- A pointer mapping edges to memory addresses: $\mathsf{lastW} : E \to \mathbb{N}$.
- A store of last values $\mathsf{deref} : \mathbb{N} \to W$.
- For each block B a set of markings $\mathsf{mark}_B \subseteq B \times \mathbb{N}$.

Notation 6.12. In the following we write $e = x \xrightarrow{a} y$ in lieu of $\mathsf{graph}(e) = (x, a, y)$.

Definition 6.13 (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) For all $x \in X, \mathsf{toSub}(x) = \varnothing$, i.e. toSub is empty everywhere.
(2) For $e_i = x_i \xrightarrow{a_i} y_i, i \in \{1, 2\}$: $\mathsf{lastW}(e_1) = \mathsf{lastW}(e_2) \iff x_1 = x_2$ and $[y_1]_{\kappa_Q} = [y_2]_{\kappa_Q}$.
(3) For every $e = x \xrightarrow{a} y$ and $C := [y]_{\kappa_Q} \in Y/Q$, we have $w(C, \xi(x)) = \mathsf{deref} \cdot \mathsf{lastW}(e)$.
(4) For every $x_1, x_2 \in B \in X/P$ and $C \in Y/Q$, we have $(x_1, x_2) \in \ker(H\chi_C \cdot \xi)$.

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 $\mathsf{graph} : E \to X \times A \times Y$ and $\mathsf{type} : X \to H1$ are implemented as arrays. In the initialization step the predecessor array $\mathsf{pred} : Y \to \mathcal{P}_\mathrm{f}E$, $\mathsf{pred}(y) = \{e \in E \mid e = x \xrightarrow{a} y\}$ is computed. Sets and bags are implemented as lists. We only insert elements into sets not yet containing them.

We say that we *group* a finite set Z by $f : Z \to Z'$ to indicate that we compute $[-]_f$. This is done by sorting the elements of $z \in Z$ 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 4.

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: $\mathsf{toSub}[x] := \emptyset$
7: $X/P :=$ group X by $\mathsf{type} : X \to H1$, $Y/Q := \{Y\}$.

Figure 4. The initialization procedure

Lemma 6.14. *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) After the procedure, for $e_i = x_i \xrightarrow{a_i} y_i, i \in \{1, 2\}$, $\mathsf{lastW}(e_1) = \mathsf{lastW}(e_2)$ iff $x_1 = x_2$ (while $[y_1]_{\kappa_Q} = [y_2]_{\kappa_Q}$ trivially holds).
(3) This is just the axiom for init in (6.1), because $Y/Q = \{Y\}$ and $\mathsf{deref} \cdot \mathsf{lastW}(e) = \mathsf{init}(\mathsf{type}(x), \mathcal{B}_\mathrm{f}\pi_1 \cdot \flat \cdot \xi(x))$.
(4) Since $\ker(H\chi_Y \cdot \xi) = \ker(H! \cdot \xi)$, this is just the way X/P is constructed. □

The algorithm for a single refinement step along a morphism $\xi : X \to HY$ is listed in Figure 5. It receives as input the two current partitions X/P and X/Q of the set of states, where the former partition is finer than the latter, a subblock $S \in X/P$ and the compound block $C \in X/Q$ it is contained in.

In the first part, all blocks $B \in X/P$ that have an edge into S are collected, together with $v_\emptyset \in H3$ which represents $H\chi_S^C \cdot \xi(x)$ 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 a pointer to $w(C, x)$.

In the second part, each block B with an edge into S is split w.r.t. $H\chi_S^C \cdot \xi$. First, for any $(x, p_C) \in \mathsf{mark}_B$, 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 distributed to new blocks w.r.t. v^x.

Lemma 6.15. *Assume that the invariants hold. Then after part (a) of Figure 5, for the given $S \subseteq C \in Y/Q$ 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* $x \in X$: $\mathsf{fil}_S \cdot \flat \cdot \xi(x) = \mathcal{B}_{\mathrm{f}}(\pi_2 \cdot \mathsf{graph})(\mathsf{toSub}(x))$.
(3) $\mathsf{M} : X/P \rightharpoonup H3$ *is a partial map defined by*

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

and $M(B)$ *is undefined otherwise.*
(4) *For each* $B \in X/P$, *we have a partial map* $\mathsf{mark}_B : B \rightharpoonup \mathbb{N}$ *defined by*

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

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* $\mathsf{fil}_S(\flat \cdot \xi(x)) = \varnothing$ *and then* $H\chi^C_S \cdot \xi(x) = H\chi^C_{\varnothing} \cdot \xi(x)$.

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

$$= |\{e \in E \mid e = x \xrightarrow{a} y, y \in S\}|.$$

(3) 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 $C = [y]_{\kappa_Q} \in Y/Q$. Then by invariant (3) we know

```
SPLIT(X/P, Y/Q, S ⊆ C ∈ Y/Q)
 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^x_C := deref · lastW[e]
 7:         v_∅ := π2 · update(∅, w^x_C)
 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^x_S, v^x, w^x_{C\S}) := update(ℓ, deref[p_C])
17:         deref[p_C] := w^x_{C\S}
18:         p_S := new cell containing w^x_S
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:     B_1 × {v_1}, ..., B_ℓ × {v_ℓ} :=
            group B_≠∅ by π2 : X × H3 → H3
25:     insert B_1, ..., B_ℓ := into X/P
```

(B) Splitting predecessor blocks

FIGURE 5. Refining X/P w.r.t $\chi^C_S : Y \to 3$ and Y/Q along $\xi : X \to HY$

that $M(B)$ is

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

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

(4) This is precisely, how mark_B has been constructed. The 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 contained 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

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

Lemma 6.16. *Given $M(B) = v_\varnothing$ in line 12. Then after line 23, $B_{\neq\varnothing}$ is a partial map $B \rightharpoonup H3$ defined by $B_{\neq\varnothing}(x) = H\chi^C_S \cdot \xi(x)$ if $H\chi^C_S \cdot \xi(x) \neq v_\varnothing$ and undefined otherwise.*

Proof. Suppose first that $x \in B$ is marked, i.e. we have $p_C = \mathsf{mark}_B(x)$ and lines 14–23 are executed. Then we have

$$\begin{aligned}(w^x_S, v^x, w^x_{C\setminus S}) &= \mathsf{update}(\mathcal{B}_\mathrm{f}(\pi_2 \cdot \mathsf{graph})(\mathsf{toSub}[x]), \mathsf{deref}[p_C]) && \text{by line 16} \\ &= \mathsf{update}(\mathsf{fil}_S \cdot \flat \cdot \xi(x), w(C, \xi(x))) && \text{by Lemma 6.15, 2 and 5} \\ &= (w(S, \xi(x)), H\chi^C_S \cdot \xi(x), w(C \setminus S, \xi(x))) && \text{by (6.1).}\end{aligned}$$

Thus, if $H\chi^C_S \cdot \xi(x) = v_\varnothing$, $B_{\neq\varnothing}$ remains undefined because of line 21, and otherwise gets correctly defined in line 23.

Now suppose that $x \in B$ is not marked. Then by Lemma 6.15(6) we know that $\mathsf{fil}_S(\flat \cdot \xi(x)) = \varnothing$. Since $(B, v_\varnothing) \in \mathsf{M}$, we know by Lemma 6.15(3) that $v_\varnothing = H\chi^C_\varnothing \cdot \xi(x')$ for some $x' \in B$. Invariant (4) then implies that $H\chi^C_\varnothing \cdot \xi(x) = H\chi^C_\varnothing \cdot \xi(x')$ since $\mathsf{ker}(H\chi^C_\varnothing \cdot \xi) = \mathsf{ker}(H\chi_C \cdot \xi)$. Then, by Lemma 6.15.6, we have

$$H\chi^C_S \cdot \xi(x) = H\chi^C_\varnothing \cdot \xi(x) = H\chi^C_\varnothing \cdot \xi(x') = v_\varnothing.$$ □

Theorem 6.17 (Correctness)**.** *If the invariants hold before invoking* Split*, then*

(i) Split *returns the correct partitions, that is,* Split$(X/P, Y/Q, S \subseteq C \in Y/Q)$ *refines X/P by $H\chi^C_S \cdot \xi : X \to H3$.*

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

Proof. (i) Lemma 6.16 shows the correctness up to line 23. So we know that all B in M are refined by $H\chi_S^C \cdot \xi$. Now let B not be in M. Then mark_B is undefined everywhere, so for all $x \in B$, we have by Lemma 6.15(6) that $H\chi_S^C \cdot \xi(x) = H\chi_\emptyset^C \cdot \xi(x)$, which is the same for all $x \in B$ by invariant (4). Hence every B not in M is not split by $H\chi_S^C \cdot \xi$. This proves the first claim.

(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 non-empty in line 11 only for marked x, and for those x it is emptied again in line 20.

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

⇒ 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 $x_1 = x_2 = x$ and $y_1, y_2 \in S \in Y/Q$. 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 $\mathsf{lastW}_{\mathsf{old}}$.

⇐ If $x_1 = x_2$ and $y_1, y_2 \in D \in Y/Q$, then we perform a case distinction on D. If $D = S$, then $\mathsf{lastW}(e_1) = \mathsf{lastW}(e_1) = p_S$. If $D = C \setminus S$, then $\mathsf{lastW}(e_1) = \mathsf{lastW}_{\mathsf{old}}(e_1) = \mathsf{lastW}_{\mathsf{old}}(e_2) = \mathsf{lastW}(e_2)$. Otherwise, $D \in Y/Q_{\mathsf{old}} \setminus \{C\}$ and again $\mathsf{lastW}(e_i) = \mathsf{lastW}_{\mathsf{old}}(e_i)$, $i \in \{1, 2\}$.

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

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

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 6.15, items 2 and 5, and by the axiom for update in (6.1) (see the computation in the proof of Lemma 6.16).

(4) Take $x_1, x_2 \in B' \in X/P$ and $D \in Y/Q$ and let $B := [x_1]_{P_{\mathsf{old}}} = [x_2]_{P_{\mathsf{old}}} \in X/P_{\mathsf{old}}$.

(a) If $\mathsf{M}(B)$ is defined, then $H\chi_S^C \cdot \xi(x_1) = H\chi_S^C \cdot \xi(x_2)$ – otherwise they would have been put into different blocks in line 24. So $(x_1, x_2) \in \ker(H\chi_D \cdot \xi(x_1))$ is obvious for $D = S$ and $D = C \setminus S$. For every other $D \in Y/Q_{\mathsf{old}}$, $(x_1, x_2) \in \ker(H\chi_D \cdot \xi)$ by the invariant of the previous partition.

(b) If $\mathsf{M}(B)$ is undefined, then mark_B is undefined everywhere, in particular for x_1 and x_2. Then, by Lemma 6.15(6) we have $H\chi_S^C \cdot \xi(x_i) = H\chi_\emptyset^C \cdot \xi(x_i)$ for $i = 1, 2$. Since $C \in Y/Q_{\mathsf{old}}$ and $\ker(H\chi_\emptyset^C \cdot \xi) = \ker(H\chi_C \cdot \xi)$, we have $H\chi_\emptyset^C \cdot \xi(x_1) = H\chi_\emptyset^C \cdot \xi(x_2)$ by invariant 4, and so $(x_1, x_2) \in \ker(H\chi_S^C \cdot \xi)$. By case distinction on D, we conclude:

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

where the last statement holds by invariant (4) for Y/Q_{old}. □

Having established correctness of Split, we can next analyse its time complexity. We first analyse lines $1 - 23$, then the complexity of the grouping operation in line 24, and finally the overall complexity of the algorithm, accumulating the time for all Split invocations.

Lemma 6.18. *Lines* $1-23$ *in* Split *run in time* $\mathcal{O}(\sum_{y\in S} |\mathsf{pred}(y)|)$.

Proof. The loop in Figure 5a 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 23 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 24, it is not enough to group the elements using a sorting algorithm. Instead we need to preprocess the elements and extract some possible majority candidate.

Definition 6.19. When grouping Z by $f : Z \to Z'$ we call an element $p \in Z'$ a *possible majority candidate* (PMC) if either

$$|\{z \in Z \mid f(z) = p\}| \geq |\{z \in Z \mid f(z) \neq p\}| \tag{6.2}$$

or if no element in Z' fulfilling (6.2) 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.

Lemma 6.20. *Summing over* Split *invocations, the total time spent on grouping* $B_{\neq\varnothing}$ *using a PMC is in* $\mathcal{O}(|E| \cdot \log |X|)$.

The proof is the same as in the weighted setting of Valmari and Franceschinis [VF10, Lemma 5]. For the convenience of the reader, we provide an adaptation to our setting:

Proof. Formally we need to prove that for a family $S_i \subseteq C_i \in Y/Q_i$, $1 \leq i \leq k$, with $Q_{i+1} = \ker\langle \kappa_{Q_i}, \chi_{S_i}^{C_i}\rangle$, the overall time spent on grouping the $B_{\neq\varnothing}$ in all the runs of Split is in $\mathcal{O}(|E| \cdot \log |X|)$.

First, we characterize the subset $\mathcal{M}_B \subseteq B_{\neq\varnothing}$ of elements that have edges into both S_i and $C_i \setminus S_i$. In the second part, we show that if we assume each sorting step of $B_{\neq\varnothing}$ is bound by $2 \cdot |\mathcal{M}_B| \cdot \log(2 \cdot |\mathcal{M}_B|)$, then the overall complexity is as desired. In the third part, we use a PMC to argue that sorting each $B_{\neq\varnothing}$ is indeed bounded as assumed. 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.

(1) For a $(B, v_\varnothing) \in \mathsf{M}$ in the ith iteration consider $B_{\neq\varnothing}$. We define
   - the *left block* $\mathcal{L}_B^i := \{x \in B \mid H\chi_{C_i}^{C_i} \cdot \xi(x) = H\chi_{S_i}^{C_i} \cdot \xi(x) \neq H\chi_\varnothing^{C_i} \cdot \xi(x)\}$, and
   - the *middle block* $\mathcal{M}_B^i := \{x \in B \mid H\chi_{C_i}^{C_i} \cdot \xi(x) \neq H\chi_{S_i}^{C_i} \cdot \xi(x) \neq H\chi_\varnothing^{C_i} \cdot \xi(x)\}$.

   Since $B_{\neq\varnothing}(x)$ is defined iff $H\chi_{S_i}^{C_i} \cdot \xi(x) \neq v_\varnothing$, and since $v_\varnothing = H\chi_\varnothing^{C} \cdot \xi(x)$ holds by Lemma 6.15(3), the domain of $B_{\neq\varnothing}$ is $\mathcal{L}_B^i \cup \mathcal{M}_B^i$. If $x \in B$ has no edge to S_i, then it is not marked, and so $H\chi_{S_i}^{C_i} \cdot \xi(x) = H\chi_\varnothing^{C} \cdot \xi(x)$, by Lemma 6.15(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 $H\chi^{C_i}_{C_i} \cdot \xi$. If $x \in B$ has no edge to $C_i \setminus S_i$, then $\mathsf{fil}_{S_i}(\flat \cdot \xi(x)) = \mathcal{B}_{\mathrm{f}}\pi_1 \cdot \flat \cdot \xi(x)$ by the definition of fil_S, and therefore we have:

$$H\chi^{C_i}_{C_i} \cdot \xi(x) \overset{(6.1)}{=} \pi_2 \cdot \mathsf{update}(\mathsf{fil}_{C_i}(\flat \cdot \xi(x)), w(C_i)) \overset{\text{Def.}}{=} \pi_2 \cdot \mathsf{update}(\mathcal{B}_{\mathrm{f}}\pi_1 \cdot \flat \cdot \xi(x), w(C_i))$$
$$= \pi_2 \cdot \mathsf{update}(\mathsf{fil}_{S_i}(\flat \cdot \xi(x)), w(C_i)) \overset{(6.1)}{=} H\chi^{C_i}_{S_i} \cdot \xi(x).$$

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

Note that $\ker(H\chi^{C_i}_{C_i} \cdot \xi) = \ker(H\chi_{C_i} \cdot \xi)$. Since $\mathcal{L}^i_B \subseteq B \in X/P_i$ and $C_i \in Y/Q_i$, we conclude from invariant (4) that there is an $\ell^i_B \in H3$ such that $\ell^i_B = H\chi^{C_i}_{C_i} \cdot \xi(x)$ for all $x \in \mathcal{L}^i_B$. Hence, we have obtained $\ell^i_B \in H3$ such that

$$\mathcal{L}^i_B = \{x \in B_{\neq \varnothing} \mid H\chi^{C_i}_{S_i} \cdot \xi(x) = \ell^i_B\} \quad \text{and} \quad \mathcal{M}^i_B = \{x \in B_{\neq \varnothing} \mid H\chi^{C_i}_{S_i} \cdot \xi(x) \neq \ell^i_B\}.$$

(2) Let the number of blocks to which x has an edge be denoted by

$$\sharp^i_Q(x) = |\{D \in Y/Q_i \mid e = x \xrightarrow{a} y, y \in D\}|.$$

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

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

Define

$$\sharp^i_{\mathcal{M}}(x) = |\{0 \leq j \leq i \mid x \text{ is in some } \mathcal{M}^j_B\}|.$$

If in the ith iteration x is in the middle block $\mathcal{M}^i_B$, then $\sharp^{i+1}_Q(x) = \sharp^i_Q(x) + 1$, since x has both an edge to S_i and $C_i \setminus S_i$ and $Q_{i+1} = \ker\langle \kappa_{Q_i}, \chi^{C_i}_{S_i} \rangle$. It follows that for all i, $\sharp^i_{\mathcal{M}}(x) \leq \sharp^i_Q(x)$, and

$$\sum_{x \in X} \sharp^i_{\mathcal{M}}(x) \leq \sum_{x \in X} \sharp^i_Q(x) \leq |E|.$$

Let T denote the total number of middle blocks $\mathcal{M}^i_B$, $1 \leq i \leq k$, B in the ith M, and let $\mathcal{M}_t$, $1 \leq t \leq T$, be the tth 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.

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

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

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

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

- If ℓ^i_B is the possible majority candidate, then the sorting of $B_{\neq \varnothing}$ sorts precisely $\mathcal{M}^i_B$ which indeed amounts to

$$|\mathcal{M}^i_B| \cdot \log(|\mathcal{M}^i_B|) \leq 2 \cdot |\mathcal{M}^i_B| \cdot \log(2 \cdot |\mathcal{M}^i_B|).$$

- 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

$$(|\mathcal{L}_B^i| + |\mathcal{M}_B^i|) \cdot \log(|\mathcal{L}_B^i| + |\mathcal{M}_B^i|) \leq 2 \cdot |\mathcal{M}_B^i| \cdot \log(2 \cdot |\mathcal{M}_B^i|). \qquad \square$$

Lemma 6.21. *For $S_i \subseteq C_i \in Y/Q_i, 0 \leq i < k$, with $2 \cdot |S_i| \leq |C_i|$ and $Q_{i+1} = \mathsf{ker}\langle \kappa_{Q_i}, \chi_{S_i}^{C_i} \rangle$:*
(1) *For each $y \in Y$, $|\{i < k \mid y \in S_i\}| \leq \log_2 |Y|$.*
(2) $\textsc{Split}(S_i \subseteq C_i \in Y/Q_i)$ *for all $0 \leq i < k$ takes at most $\mathcal{O}(|E| \cdot \log |Y|)$ time in total.*

Proof. (1) We know from Lemma 4.6 that Q_{i+1} is finer than Q_i. Moreover, since $\chi_{S_i}^{C_i}$ merges all elements of S_i we have $S_i \in Y/Q_{i+1}$. 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 Y/Q_j of Y/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 $2^n \cdot |S_{i_n}| \leq |S_{i_1}|$. Thus

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

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

(2) In the $\mathcal{O}$-calculus the total time complexity is:

$$\begin{aligned}
\sum_{0 \leq i < k} \sum_{y \in S_i} |\mathsf{pred}(y)| &= \sum_{y \in Y} \sum_{\substack{0 \leq i < k \\ S_i \ni y}} |\mathsf{pred}(y)| = \sum_{y \in Y} \big(|\mathsf{pred}(y)| \cdot \sum_{\substack{0 \leq i < k \\ S_i \ni y}} 1 \big) \\
&\leq \sum_{y \in Y} \big(|\mathsf{pred}(y)| \cdot \log |Y| \big) = \big(\sum_{y \in Y} |\mathsf{pred}(y)| \big) \cdot \log |Y| \\
&= |E| \cdot \log |Y|,
\end{aligned}$$

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

Bringing Sections 4, 5, and 6 together, take a coalgebra $\xi : X \to HX$ for a zippable Set-functor H with a refinement interface such that init and update run in linear and comparisons in constant time. Instantiate Algorithm 4.5 with the select routine from Example 4.4.1, chosing $q_{i+1} = k_{i+1} \cdot \kappa_{P_i} = \chi_{S_i}^{C_i}$ with $2 \cdot |S_i| \leq |C_i|$, $S_i, C_i \subseteq X$ and replace line 4 by

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

By Theorem 6.17 this is equivalent to (5.1), and by Corollary 5.18 equivalent to the original line 4, since $\chi_{S_i}^{C_i}$ respects compound blocks. By Lemmas 6.14 and 6.21(2), we have

Theorem 6.22. *The above instance of Algorithm 4.5 computes the quotient modulo behavioural equivalence of a given coalgebra in time $\mathcal{O}((m+n) \cdot \log n)$, for $m = |E|$, $n = |X|$.*

If the coalgebra is not too sparse, i.e. every state has at least one in- or outgoing edge, $m \geq 2 \cdot n$, 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 $\xi \colon X \to \mathbb{R}^{(X)}$. However, 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 4.10) we can equip the generic algorithm with this additional parameter while maintaining the same $\mathcal{O}(m \cdot \log n)$ complexity:

Remark 6.23. There are two ways to handle functors of type $H = X/\mathcal{I} \times G$. First, we can modify the functor interface as follows:

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

update is replaced by the following

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

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 7 for more details, in particular Example 7.17. Both methods have no effect on the complexity.

Example 6.24. 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 $\mathcal{O}((m+n) \cdot \log n)$.
(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 (Valmari and Franceschinis [VF10]).
(3) Hopcroft's classical automata minimization [Hop71] is obtained by $HX = 2 \times X^A$, with running time $\mathcal{O}(n \cdot \log n)$ for a fixed alphabet A. If A is variable and part of the input, we can decompose the given coalgebra into a multisorted one (see Example 7.17 below).

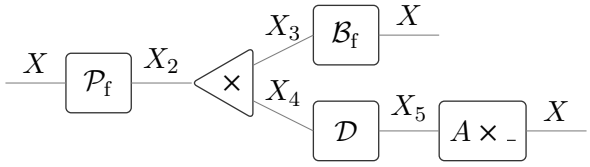
7. Modularity via Multisorted Coalgebra

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.

7.1. Explicit intermediate states via multisortedness.

The transformation from coalgebras for composite functors into multisorted coalgebras is best understood by example:

Example 7.1. 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 $\bar{T}\colon \mathsf{Set}^5 \to \mathsf{Set}^5$:

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

Formal definitions following [SP11] are as follows.

Definition 7.2. 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

$$G ::= (\_) \mid H(G, \dots, G)$$

where H ranges over $\mathcal{H}$. Such a functor T can be converted into 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. 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 7.1).

Example 7.3. In Example 7.1 the functor T is built from the set $\mathcal{H}$ of functors containing

$$\mathcal{P}_\mathrm{f}, \mathcal{B}_\mathrm{f}, \mathcal{D}, A \times \_\colon \mathsf{Set}^1 \to \mathsf{Set} \qquad \times\colon \mathsf{Set}^2 \to \mathsf{Set},$$

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:

$$\begin{array}{lll} 1.\ \mathcal{P}_\mathrm{f}(\mathcal{B}_\mathrm{f}(\_) \times \mathcal{D}(A \times (\_))) & 2.\ \mathcal{B}_\mathrm{f}(\_) \times \mathcal{D}(A \times (\_)) & 3.\ \mathcal{B}_\mathrm{f}(\_) \\ 4.\ \mathcal{D}(A \times (\_)) & 5.\ A \times (\_). & \end{array}$$

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 given a T-coalgebra $\xi\colon X \to TX$, the morphism $(\xi, \mathsf{id}, \dots, \mathsf{id})$ 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].

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

The left adjoint Comp composes the component maps of a multisorted coalgebra on $(X_1, \dots, X_n)$ in a suitable way to obtain a T-coalgebra on X_1, and so $\mathsf{Comp}(\mathsf{Pad}(X, \xi)) = (X, \xi)$. On morphisms, one has $\mathsf{Comp}(h_1, \dots, h_n) = h_1$.

Example 7.4. 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 7.3, we obtain the $\bar{T}$-coalgebra

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

However, this coalgebra is not finite anymore, because e.g. $\mathcal{B}_\mathrm{f}(X)$ is infinite for non-empty X. It is possible to find a finite $\bar{T}$-coalgebra that conforms to ξ by restricting e.g. the sort $\mathcal{B}_\mathrm{f}(X)$ do 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 $f\colon X \to H(Y_1,\dots,Y_k)$ there exist finite subsets $m_i : Y_i' \rightarrowtail Y_i$ such that f factors through $H(m_1,\dots,m_k)$:

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

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 7.5. Let $\mathcal{H}$ be a set of finitary functors, let $T : \mathsf{Set} \to \mathsf{Set}$ be a functor as in Definition 7.2 with flattening $\bar{T}$, and let (X,ξ) be a T-coalgebra. We construct a $\bar{T}$-coalgebra $\mathsf{Factor}(X,\xi)$ by repeatedly applying the above factorization technique: if $T = H(G_1,\dots,G_k)$ choose finite subsets $m_i : Y_i \rightarrowtail G_i X$, for $i = 1,\dots,k$ as small as possible, and a map $\xi' : X \to H(Y_1,\dots,Y_n)$ such that $H(m_1,\dots,m_n)\cdot\xi' = \xi$. Then the structure map in the first sort of $\mathsf{Factor}(X,\xi)$ is ξ', and one continues recursively to factorize the m_i. Note that the $\bar{T}$-coalgebra thus obtained has all sorts finite, it satisfies $(X,\xi) = \mathsf{Comp}(\mathsf{Factor}(X,\xi))$, and

$$\mathsf{Factor}(X,\xi) \xrightarrow{(\mathsf{id}_X, m_1, m_2, \dots)} \mathsf{Pad}(X,\xi)$$

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

Remark 7.6. We do not need that Factor is functorial here. 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 7.7. Consider a finite coalgebra $\xi\colon X \to TX$ for $T = \mathcal{P}_\mathrm{f}(\mathcal{B}_\mathrm{f}(\_) \times \mathcal{D}(A \times (\_)))$ from Example 7.3. The above Construction 7.5 yields a (multisorted) $\bar{T}$-coalgebra $\mathsf{Factor}(X,\xi)$ with finite carriers (X, Y_1, Y_2, Y_3, Y_4), and structure maps

$$\xi'\colon X \to \mathcal{P}_\mathrm{f} Y_1 \quad y_1\colon Y_1 \to Y_2 \times Y_3 \quad y_2\colon Y_2 \to \mathcal{B}_\mathrm{f} Z \quad y_3\colon Y_3 \to \mathcal{D} Y_4 \quad y_4\colon Y_4 \to A \times Z.$$

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

Proof. From every T-coalgebra morphism $q\colon \mathsf{Comp}(\bar{X},\bar{\xi}) \to (Y,\zeta)$ we can construct a $\bar{T}$-coalgebra morphism q' from $(\bar{X},\bar{\xi})$ with first component q. Such a morphism is obtained as

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

Since $(\bar{X},\bar{\xi})$ is simple, q' must be a mono in Set^n, thus q is an injective map, which shows that $\mathsf{Comp}(\bar{X},\bar{\xi})$ is simple, too (see Proposition 2.8). □

Corollary 7.9. *If* $q\colon \mathsf{Factor}(X,\xi) \twoheadrightarrow (Y,\zeta)$ *represents the simple quotient of a* T*-coalgebra* (X,ξ)*, 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 place 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 7.3.

7.2. **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 modeling 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 7.10. 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.

In this notation, we can define extensivity of $\mathcal{C}$ in the following way:

Definition 7.11. A category $\mathcal{C}$ is *extensive* if the following equivalence holds for every $n \geq 0$ and every commuting square in $\mathcal{C}^n$ as below: the square is a pullback iff $[f]\colon \coprod X' \to X$ is an isomorphism.

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

Remark 7.12. The usual definition of extensivity states that the canonical functor $\mathcal{C}/A \times \mathcal{C}/B \to \mathcal{C}/A + B$ is an equivalence of categories for every pair A, B of objects. Our Definition 7.11 above is easily seen to be equivalent to the characterization of extensivity in terms of pullbacks of coproducts given in [CLW93, Proposition 2.2]. In every extensive category, coproduct injections are monic, so $\eta_X\colon X \rightarrowtail \Delta\coprod X$ is a mono.

Example 7.13. 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.

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

(1) The categories of $\bigsqcup H\Delta$-coalgebras and $H\Delta\bigsqcup$-coalgebras are equivalent (Lemma 7.14).
(2) The obvious functor from H-coalgebras to $H\Delta\bigsqcup$-coalgebras given by

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

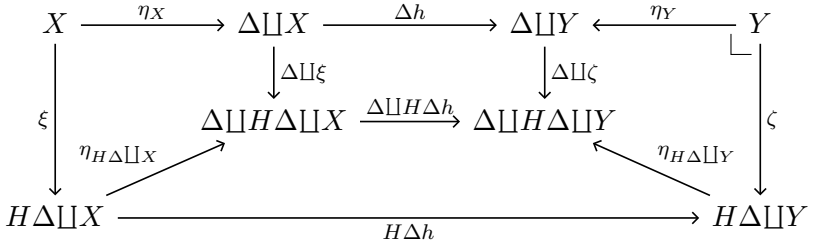
preserves and reflects (simple) quotients (Lemma 7.15).

Lemma 7.14. *The lifting* $\bar{\bigsqcup}$ *of the coproduct functor* $\bigsqcup$

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

is an equivalence of categories.

Proof. We have to show that $\bar{\bigsqcup}$ is full, faithful and isomorphism-dense [AHS90]. Faithfulness is immediate from the fact that already $\bigsqcup : \mathcal{C}^n \to \mathcal{C}$ is faithful, since coproduct injections are monic (Remark 7.12). To see that $\bar{\bigsqcup}$ is full, let $h\colon \bar{\bigsqcup}(X,\xi) \to \bar{\bigsqcup}(Y,\zeta)$ be a $\bigsqcup H\Delta$-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 $h' : X \to Y$ such that $\zeta \cdot h' = H\Delta h \cdot \xi$ and $\eta_Y \cdot h' = \Delta h \cdot \eta_X$. The latter equality implies $\bigsqcup h' = h$ using the universal property of η_X, and then the first equality states that $h' : (X,\xi) \to (Y,\zeta)$ is an $H\Delta\bigsqcup$-coalgebra morphism.

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

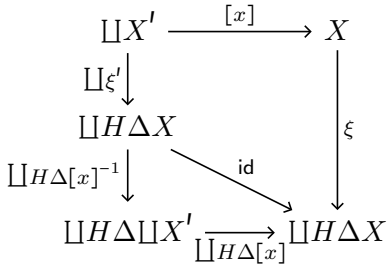
$$\begin{array}{ccc} X' & \xrightarrow{x} & \Delta X \\ {\scriptstyle \xi'}\downarrow & \lrcorner & \downarrow{\scriptstyle \Delta\xi} \\ H\Delta X & \xrightarrow{\eta_{H\Delta X}} & \Delta\bigsqcup H\Delta X \end{array} \tag{7.1}$$

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

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

Applying the adjunction to the square (7.1) shows $\xi \cdot [x] = \bigsqcup\xi'$. It follows that the isomorphism $[x]$ is a coalgebra morphism, and hence an isomorphism in $\mathsf{Coalg}(\bigsqcup H\Delta)$, from

$\bar{\coprod}(X', H\Delta[x]^{-1} \cdot \xi')$ to (X, ξ):



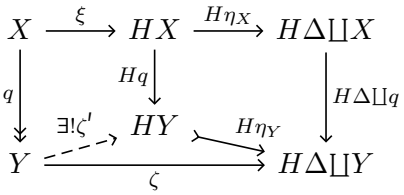
□

Lemma 7.15. *Any H-coalgebra $\xi\colon X \to HX$ and the induced $H\Delta\coprod$-coalgebra*

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

have the same quotients and the same simple quotients.

Proof. Let $q : X \to Y$ be a regular epimorphism. It suffices to show that q carries an H-coalgebra morphism with domain (X, ξ) iff it carries an $H\Delta\coprod$-coalgebra morphism with domain $(X, H\eta_X \cdot \xi)$. Since η induces an embedding $\mathsf{Coalg}(H) \to \mathsf{Coalg}(H\Delta\coprod)$, 'only if' is clear; we prove 'if'. So suppose that q is a coalgebra morphism $(X, H\eta \cdot \xi) \to (Y, \zeta)$. Then the outside of the following diagram commutes:



Since η_X is monic and H preserves monos, $H\eta_Y$ is monic, so we obtain ζ' as in the diagram by the diagonal property (Section 2.1), making q an H-coalgebra morphism $(X, \xi) \to (Y, \zeta')$. □

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 6.7.

7.3. **Coproducts of refinement interfaces.** We have already seen that zippable functors are closed under coproducts (Lemma 5.4). 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 filtering a set of labels for a particular 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 somewhat from fil_S as in (6.1)). Then we implement the refinement interface for H component-wise as follows (writing $I = \{1, \dots, n\}$):

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

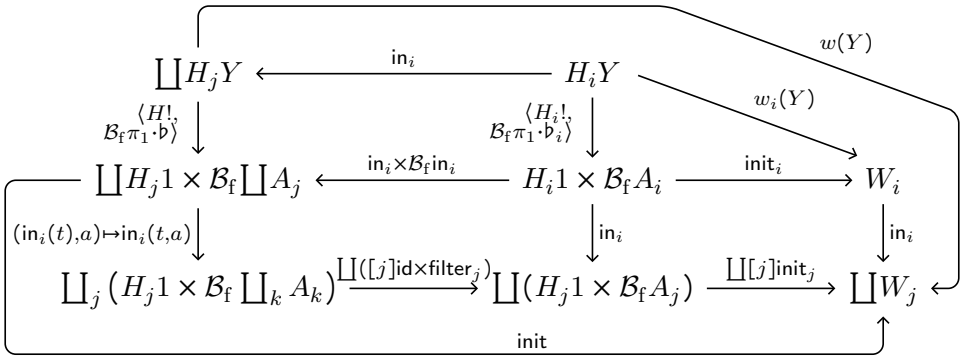
$$w(S) = \Big(\coprod H_i Y \xrightarrow{\coprod w_i(S)} \coprod W_i\Big) \qquad \text{for } S \subseteq Y,$$

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

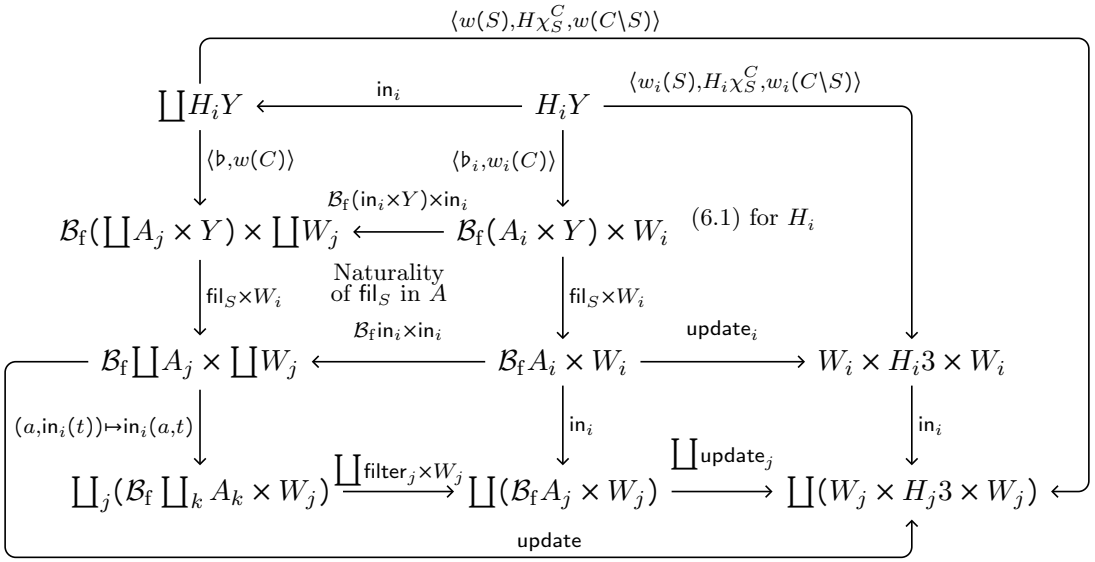
$$\mathsf{update} = \Big(\mathcal{B}_\mathrm{f}\coprod A_i \times \coprod W_i \to \coprod_i \big(\mathcal{B}_\mathrm{f}\coprod_j A_j \times W_i\big) \to \coprod (\mathcal{B}_\mathrm{f} A_i \times W_i) \longrightarrow \coprod (W_i \times H_i 3 \times W_i) \downarrow \coprod W_i \times \coprod H_i 3 \times \coprod W_i\Big)$$

Proposition 7.16. *The data W, A, b, 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 6.7, then so does the one of H.*

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



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



Using again that the $\mathsf{in}_i \colon H_i Y \to \coprod H_i$ are jointly epic, we see that the claimed refinement interface fulfils the axiom for update. If for every i, the interface of H_i fulfils the time constraints from Assumption 6.7, 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_i3. 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 7.1 and 7.2, 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 6.24). We build our examples from the functors

$$\mathcal{D}, \mathcal{P}_\mathrm{f}, A \times \_ \colon \mathsf{Set}^1 \to \mathsf{Set} \qquad \times, + \colon \mathsf{Set}^2 \to \mathsf{Set},$$

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

Example 7.17. (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

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

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 6.24).

(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, i.e. as coalgebras $\xi \colon X \to 2 \times \mathcal{P}_\mathrm{f}(\mathbb{N} \times X)$. 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 $m \colon A \hookrightarrow \mathbb{N}$ is then represented by the two-sorted system

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

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

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

(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 7.16. 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 $\coprod \bar{H}\Delta$-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 $\xi : X \to \mathcal{P}_\mathrm{f}(A \times \mathcal{D}X)$, Baier, Engelen, Majster-Cederbaum [BEM00] 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

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

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 $\mathcal{O}((n + m_p + m_d) \cdot \log(n + m_p))$. Since $m_d \leq n \cdot m_p$, this matches the run time $\mathcal{O}((n \cdot m_p) \cdot (\log n + \log m_p))) = \mathcal{O}((n \cdot m_p) \cdot (\log(n + m_p)))$ of the best previous algorithm [BEM00], and indeed provides a more fine-grained analysis giving faster run time in the (presumably wide-spread) case that probabilistic transitions are sparse, i.e. if m_d is substantially below $n \cdot m_p$. In other words, our bound $\mathcal{O}((n + m_p + m_d) \cdot \log(n + m_p))$ is smaller if the number of all (non-deterministic and probabilistic) transitions is taken into account, rather than just the non-deterministic transitions.

(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, resulting in a run time in $\mathcal{O}((m_p + m_d) \cdot \log(n + m_p + m_d))$.

We are not aware of any pre-existing similarly efficient partition refinement algorithms for alternating systems (Example 7.17(3)) and general Segala systems (Example 7.17(5))

8. Conclusions and Future 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], and Valmari and Franceschinis's lumping algorithm for weighted transition systems [VF10]. Moreover, we

presented a generic method apply the algorithm to mixed system types. As an instance, we obtain a new algorithm for simple Segala systems that allows for a more fine-grained analysis of asymptotic run time than previous algorithms [BEM00].

It remains open whether our approach can be extended to, e.g., the monotone neighbourhood functor, which is not itself zippable (see Example 5.8) 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.

References

[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.

[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.

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

[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.

[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, *Proc. 28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl, 2017.

[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.

[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.

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

[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.

[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.

[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.

[LM92] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic*. Springer New York, 1992.

[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.

[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.

[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.

[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.