# Explicit Hopcroft's Trick in Categorical Partition Refinement

**Takahiro Sanada**
RIMS, Kyoto University, Japan

**Ryota Kojima**
RIMS, Kyoto University, Japan

**Yuichi Komorida**
National Institute of Informatics & SOKENDAI, Tokyo, Japan

**Koko Muroya**
RIMS, Kyoto University, Japan

**Ichiro Hasuo**
National Institute of Informatics & SOKENDAI, Tokyo, Japan



## Abstract

Algorithms for *partition refinement* are actively studied for a variety of systems, often with the optimisation called *Hopcroft's trick*. However, the low-level description of those algorithms in the literature often obscures the essence of Hopcroft's trick. Our contribution is twofold. Firstly, we present a novel formulation of Hopcroft's trick in terms of general trees with weights. This clean and explicit formulation—we call it *Hopcroft's inequality*—is crucially used in our second contribution, namely a general partition refinement algorithm that is *functor-generic* (i.e. it works for a variety of systems such as (non-)deterministic automata and Markov chains). Here we build on recent works on coalgebraic partition refinement but depart from them with the use of *fibrations*. In particular, our fibrational notion of *R-partitioning* exposes a concrete tree structure to which Hopcroft's inequality readily applies. It is notable that our fibrational framework accommodates such algorithmic analysis on the categorical level of abstraction.

**2012 ACM Subject Classification** Theory of computation

**Keywords and phrases** partition refinement, category theory, coalgebra, fibration, tree algorithm

**Acknowledgements** The authors are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST. Y.K. is supported by JSPS KAKENHI Grant Number JP21J13334. T.S. is supported by JST Grant Number JPMJFS2123. R.K. is supported by JST Grant Number JPMJFS2123.

## 1 Introduction

*Partition refinement* refers to a class of algorithms that computes behavioural equivalence of various types of systems—such as the language equivalence for deterministic finite automata (DFAs), bisimilarity for labelled transition systems (LTSs) and Markov chains, etc.—by a fixed-point iteration. Such algorithms also yield *quotients* of state spaces, making systems smaller and thus easier to analyse e.g. by model checking.

Since its original introduction by Moore [20] for DFAs, partition refinement has been actively studied for enhanced performance and generality. On the performance side, Hopcroft [11] introduced what is now called *Hopcroft's trick* that greatly improves the asymptotic complexity. The original paper [11] is famously hard to crack; works such as [6, 16] present its reformulation, again focusing on DFAs. On the generality side, partition refinement for systems other than DFAs has been pursued, such as LTSs [15,24], probabilistic transition systems with non-determinism [7], weighted automata [18], and weighted tree automata [9, 10]. Hopcroft's trick is used in many of these works for enhanced performance, too.

Such a variety of target systems is uniformly addressed by a recent body of work on *coalgebraic partition refinement* [4, 5, 14, 26]. Here, a target system is identified with a categorical construct called *coalgebra* $c\colon C \to FC$ (see e.g. [13]), where $C$ represents the state space, the *functor* $F$ specifies the type of the system, and $c$ represents the dynamics. By changing the functor $F$ as a parameter, the theory accommodates many different systems such as DFAs and weighted automata. The coalgebraic partition refinement algorithms in [4, 5, 14, 26] are *functor-generic*: they apply uniformly to such a variety of systems.

The current work is inspired by [14] which successfully exploits Hopcroft's trick for generic coalgebraic partition refinement. In [14], their coalgebraic algorithm is described in parallel with its set-theoretic (or even binary-level) concrete representations, letting the latter accommodate Hopcroft's trick. Their experimental results witnessed its superior performance, beating some existing tools that are specialised in a single type of systems.

However, the use of Hopcroft's trick in [14] is formulated in low-level set-theoretic terms, which seems to obscure the essence of the algorithm as well as the optimisation by Hopcroft's trick, much like in the original paper [11]. Therefore, in this paper, we aim at 1) an explicit formulation of Hopcroft's trick, and 2) a categorical partition refinement algorithm that exposes an explicit data structure to which Hopcroft's trick applies.

We achieve these two goals in this paper: 1) an explicit formulation that we call *Hopcroft's inequality*, and 2) a categorical algorithm that uses a *fibration*. Here is an overview.

**Hopcroft's Inequality** We identify *Hopcroft's inequality* (Thm. 2.9) as the essence of Hopcroft's trick. Working on general trees with a general notion of vertex weight, it uses the classification of edges into *heavy* and *light* ones and bounds a sum of weights in terms of (only) the root and leaf weights. This inequality can be used to bound the complexity of many tree generation algorithms, including those for partition refinement.

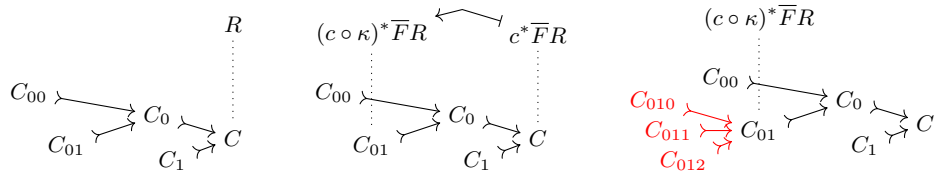
This general theory can accommodate different weights. We exploit this generality to systematically derive partition refinement algorithms with different complexities (§6.2).

**A Fibrational Partition Refinement Algorithm** Hopcroft's inequality does not directly apply to the existing coalgebraic partition refinement algorithms [4, 5, 14, 26] since the latter do not explicitly present a suitable tree structure. To address this challenge, we found the categorical language of *fibrations* [12] to be a convenient vehicle: it allows us to speak about the relationship between 1) an equivalence relation (an object in a fibre category) and 2) a partitioning of a state space (a mono-sink in the base category). The outcome is a partition refinement algorithm that is both *abstract* (it is functor-generic and applies to a variety of systems) and *concrete* (it explicitly builds a tree to which Hopcroft's inequality applies.) Our development relies on the fibrational theory of bisimilarity [8, 17]; yet ours is the first fibrational partition refinement algorithm.

More specifically, in a fibration $p\colon \mathbb{E} \to \mathbb{C}$, an equivalence relation $R$ on a set $X$ is identified with an object $R \in \mathbb{E}_X$ in the fibre over $X$ (consider the well-known fibration $\mathbf{EqRel} \to \mathbf{Set}$ of sets and equivalence relations over them). We introduce a categorical notion of *$R$-partitioning*; it allows $R \in \mathbb{E}_X$ to induce a mono-sink (i.e. a family of monomorphisms) $\{\kappa_i\colon C_i \rightarrowtail C\}_{i\in I}$. The latter is identified with the set of $R$-equivalence classes.

Fig. 1 illustrates one iteration of our fibrational partition refinement algorithm $\mathsf{fPR}^{\mathrm{H}}$ (Algo. 2). In the last step (Fig. 1c), the mono-sink $C_{010}, C_{011}, C_{012} \rightarrowtail C_{01}$ arises as the $(c\circ\kappa)^*\overline{F}R$-partitioning of $C_{01}$. In this manner, a tree structure explicitly emerges in the base category $\mathbb{C}$. Hopcroft's inequality directly applies to this tree, allowing us to systematically present the Hopcroft-type optimisation on the categorical level of abstraction.

We note that, at this moment, our fibrational framework (with a fibration $p\colon \mathbb{E} \to \mathbb{C}$) has only one example, namely the fibration $\mathbf{EqRel} \to \mathbf{Set}$ of equivalence relations over sets.




**(a)** Before the iteration **(b)** Refine $R$ and restrict to $C_{01}$ **(c)** Refine $C_{01}$ and expand the tree

**Figure 1** An iteration in our algorithm $\mathsf{fPR}^{\mathrm{H}}$ (Algo. 2). Fig. 1a shows an equivalence relation $R$ over $C$, and the corresponding partitioning $C_{00}, C_{01}, C_1 \rightarrowtail C$ of the state space $C$. (The history of refinement is recorded as a tree; this is important for complexity analysis.) In Fig. 1b, the equivalence relation $R$ is refined into $c^*\overline{F}R$ along the one-step transition of the system dynamics $c$, and is further restricted to the partition $C_{01}$. In Fig. 1c, the resulting equivalence relation $(c \circ \kappa)^* \overline{F} R$ over $C_{01}$ yields a partitioning of $C_{01}$, expanding the tree

While it is certainly desirable to have other examples, their absence does not harm the value of our fibrational framework: we do not use fibrations for additional generality (beyond functor-genericity);[1] we use them to explicate trees in the base category (cf. Fig. 1).

**Contributions** Summarising, our main technical contributions are as follows.

- *Hopcroft's inequality* that explicates the essence of Hopcroft's trick.
- A fibrational notion of *$R$-partitioning* that turns a fibre object into a mono-sink (§4).
- A fibrational partition refinement algorithm $\mathsf{fPR}^{\mathrm{H}}$ that combines the above two (§6.1).
- Functor-generic partition refinement algorithms $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{C}}}, \mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{P}}}, \mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{R}}}$, obtained as instances of $\mathsf{fPR}^{\mathrm{H}}$ but using different weights in Hopcroft's inequality. The three achieve slightly different, yet comparable to the best known, complexity bounds (§6.2).

## 2 Hopcroft's Inequality

We present our first contribution, *Hopcroft's inequality*. It is a novel formalisation of Hopcroft's trick in terms of rooted trees. It also generalises the trick, accommodating arbitrary *weights* (Def. 2.2) besides the particular one that is typically and widely used (e.g. [11, 14, 15, 24]).

▶ **Notation 2.1.** Let $T$ be a rooted tree. We denote the set of leaves by $L(T)$, the set of vertices by $V(T)$, the set of edges in the path from $v$ to $u$ by $\mathrm{path}(v, u)$, the set of children of $v \in V(T)$ by $\mathrm{ch}(v)$, and the subtree whose root is $v \in V(T)$ by $\mathrm{tr}(v)$.

▶ **Definition 2.2** (weight function)**.** Let $T$ be a rooted finite tree. A *weight function* of $T$ is a map $w \colon V(T) \to \mathbb{N}$ satisfying $\sum_{u \in \mathrm{ch}(v)} w(u) \leq w(v)$ for each $v \in V(T)$. We call a weight function *tight* if $\sum_{u \in \mathrm{ch}(v)} w(u) = w(v)$ for all $v \in V(T) \setminus L(T)$.

▶ **Definition 2.3** (heavy child choice)**.** For a weight function $w$ of a tree $T$, a *heavy child choice* (hcc for short) is a map $h \colon V(T) \setminus L(T) \to V(T)$ satisfying $h(v) \in \mathrm{ch}(v)$ and $w(h(v)) = \max_{u \in \mathrm{ch}(v)} w(u)$ for every $v \in V(T) \setminus L(T)$. We write $h(v)$ as $h_v$ and call the vertex $h_v$ a *heavy child* of $v$, and a non-heavy child a *light child*. We define $\mathrm{lch}_h(v) = \mathrm{ch}(v) \setminus \{h_v\}$. An edge $(v, u)$ is a *light edge* if $u \in \mathrm{lch}_h(v)$. We define $\mathrm{lpath}(v, u) = \{e \in \mathrm{path}(v, u) \mid e \text{ is a light edge}\}$.

Note that a heavy child choice always exists but is not unique in general.

[1] In this sense, we can say that our use of fibrations is similar to some recent usages of string diagrams in *specific* monoidal categories, such as in [2, 22].

Examples are in Fig. 2; the weight on the left is not tight while the right one is tight.

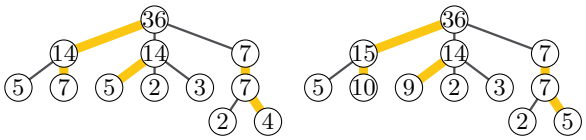



**Figure 2** Examples of rooted trees, each with a weight function and an hcc. The heavy children are indicated by thick edges. A thin edge represents a light edge.

In the rest of this section, our technical development is towards Hopcroft's inequality in Thm. 2.9. It gives an upper bound for a sum of weights—we only count those for light children, which is the core of the optimisation in Hopcroft's partition refinement algorithm [11]—in terms of weights of the root and the leaves. This upper bound makes no reference to the tree's height or internal weights, making it useful for complexity analysis of tree generation algorithms.

The following lemma crucially relies on the definition of weight function.

▶ **Lemma 2.4.** *Let $T$ be a finite tree with a root $r$, $w$ be a weight function of $T$, and $S$ be an arbitrary set of edges of $T$. Then* $\sum_{v\in V(T)} \sum_{\substack{u\in \mathrm{ch}(v)\\(v,u)\notin S}} w(u) \geq \sum_{l\in L(T)} \big|\mathrm{path}(r,l)\setminus S\big| \cdot w(l)$ *holds. The equality holds when $w$ is tight.*

Lem. 2.5 is our first key lemma; we use Lem. 2.4 in its proof. It relates the sum of weights of the light children—for which we aim to give an upper bound in Thm. 2.9—with the leaf weights and (roughly) the tree height.

▶ **Lemma 2.5.** *Let $T$ be a finite tree with a root $r$, $w$ be a weight function of $T$, and $h$ be an hcc for $w$. Then the following inequality holds. The equality holds when $w$ is tight.*

$$\sum_{v\in V(T)} \sum_{u\in \mathrm{lch}_h(v)} w(u) \;\geq\; \sum_{l\in L(T)} \big|\mathrm{lpath}(r,l)\big| \cdot w(l). \tag{1}$$

For the right tree in Fig. 2, the left-hand side of (1) is $(14+7)+5+(2+3)+0+2=33$, and the right-hand side is $1\times 5+0\times 10+1\times 9+2\times 2+2\times 3+2\times 2+1\times 5=33$.

The inequality in (1) is the opposite of what we want (namely an upper bound for the left-hand side). We thus force an equality using the following notion of tightening.

▶ **Definition 2.6** (tightening)**.** Let $w$ be a weight function of a rooted finite tree $T$, and $h$ be its heavy child choice. The *tightening* $w'\colon V(T)\to\mathbb{N}$ of $w$ along $h$ is defined recursively by

$$w'(u) = \begin{cases} w(u) & \text{if } u \text{ is the root of } T\\ w'(v) - \sum_{u'\in \mathrm{lch}_h(v)} w(u') & \text{if } u = h_v \text{ for the parent } v \text{ of } u\\ w(u) & \text{otherwise.}\end{cases}$$

In Fig. 2, the weight function of the right tree is a tightening of that of the left tree. We observe that tightening maintains a heavy child choice:

▶ **Lemma 2.7.** *Let $T$ be a rooted finite tree, $w$ be a weight function of $T$, $h$ be an hcc for $w$, and $w'$ be the tightening of $w$ along $h$. The following hold.*

1. *The map $w'$ is a tight weight function of $T$.*
2. *The map $h$ is also a heavy child choice for $w'$.*
3. *For the root $r$, $w(r)=w'(r)$ holds, and for each $v\in V(T)$, $w(v)\leq w'(v)$ holds.*

Our second key lemma towards Thm. 2.9 is as follows, bounding $|\,\mathrm{lpath}(r,v)|$ that occurs on the right in (1). Its proof is by what is commonly called *Hopcroft's trick* [1, 11]: it observes that, along a light edge, weights decay at least by $1/2$.

▶ **Lemma 2.8.** *Let $T$ be a finite tree with a root $r$, $w$ be a weight of $T$, and $h$ be an hcc for $w$. For each vertex $v \in V(T)$ with $w(v) \neq 0$, the following inequality holds:* $|\mathrm{lpath}(r, v)| \leq \log_2 w(r) - \log_2 w(v)$.

We combine Lem. 2.8 and Lem. 2.5 (its equality version; we can use it via tightening) to obtain Hopcroft's inequality. It bounds a sum of weights by the root and leaf weights.

▶ **Theorem 2.9** (Hopcroft's inequality)**.** *Let $T$ be a finite tree with root $r$, $w$ be a weight function of $T$, and $h$ be a heavy child choice for $w$. The following inequality holds.*

$$\sum_{v \in V(T)} \sum_{u \in \mathrm{lch}_h(v)} w(u) \;\leq\; w(r) \log_2 w(r) - \sum_{l \in L(T), w(l) \neq 0} w(l) \log_2 w(l). \tag{2}$$

For complexity analysis, we use Hopcroft's inequality in the following form. Assume that a tree generation algorithm takes $t(v)$ time to generate all the children (both heavy and light) of $v$. If there exists $K$ with $t(v) \leq K \sum_{u \in \mathrm{lch}(v)} w(u)$—note that the bound only refers to light children—then the time to generate the whole tree is bounded by $Kw(r) \log_2 w(r)$.

▶ **Corollary 2.10.** *Let $T$ be a rooted finite tree with root $r$, $w$ be a weight function of $T$, and $h$ be a heavy child choice for $w$. If a map $t \colon V(T) \to \mathbb{N}$ satisfies that there exists a constant $K \in \mathbb{N}$ such that $t(v) \leq K \sum_{u \in \mathrm{lch}_h(v)} w(u)$ for every $v \in V(T)$, then the sum of $t(v)$ is bounded by $Kw(r) \log_2 w(r)$, that is $\sum_{v \in V(T)} t(v) \leq Kw(r) \log_2 w(r)$.*

▶ **Remark 2.11.** Further adaptations of Hopcroft's trick are pursued in the literature, e.g. in [25], where the notion of heavy child choice is relaxed with an extra parameter $\alpha \in [1/2, 1)$. Our theory can easily be extended to accommodate $\alpha$, in which case the above description corresponds to the special case with $\alpha = 1/2$. Details are deferred to another venue.

## 3 Categorical Preliminaries

The rest of the paper is about our second contribution, namely a functor-generic partition refinement (PR) algorithm optimised by an explicit use of Hopcroft's inequality (Thm. 2.9). It is given by our novel formulation of coalgebraic PR algorithms in fibrational terms. Here we shall review some necessary categorical preliminaries.

We use categorical formalisation of intersections and unions for monomorphisms.

▶ **Definition 3.1.** For monomorphisms $m \colon A \rightarrowtail C$ and $n \colon B \rightarrowtail C$ in $\mathbb{C}$, the *intersection* $m \cap n \colon A \cap B \rightarrowtail C$ and the *union* $m \cup n \colon A \cup B \rightarrowtail C$ are defined by the following pullback and pushout, respectively:

$$\begin{array}{ccc} A \cap B & \xrightarrow{\pi_2} & B \\ {\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle n} \\ A & \xrightarrow{m} & C \end{array}, \text{ and } \quad \begin{array}{ccc} A \cap B & \xrightarrow{\pi_2} & B \\ {\scriptstyle \pi_1}\downarrow & & \downarrow \\ A & \longrightarrow & A \cup B \end{array}\ \ulcorner .$$

We say $m \colon A \rightarrowtail C$ and $m' \colon A' \rightarrowtail C$ are *equivalent* if there is an isomorphism $\phi \colon A \to A'$ such that $m = m' \circ \phi$. The set $\mathbf{Sub}(\mathbb{C})_C$ of equivalence classes of monomorphisms whose codomains are $C$ forms a lattice, assuming enough limits and colimits.

### 3.1 Fibrations

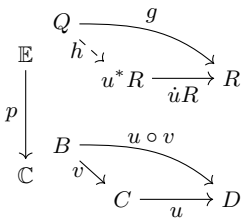
A fibration $p \colon \mathbb{E} \to \mathbb{C}$ is a functor satisfying some axioms. When $p(R) = C$ for an object $R \in \mathbb{E}$ and an object $C \in \mathbb{C}$, we see that $R$ equips $C$ with some information, e.g. a predicate, a relation, a topology, etc. The main example in this paper is the fibration $\mathbf{EqRel} \to \mathbf{Set}$ where $C$ is a set and $R$ is an equivalence relation over $C$.

Fibrational constructs that are the most relevant to us are the *inverse image* $f^*(R')$ and the *direct image* $f_*(R)$ along a morphism $f\colon S \to S'$ in $\mathbb{C}$. In the case of **EqRel** $\to$ **Set**, these are computed as follows.


In what follows we introduce some basics of fibrations; they formalise the intuition above. For details, see e.g. [12].

▶ **Definition 3.2** (fibration)**.** Let $p\colon \mathbb{E} \to \mathbb{C}$ be a functor. A morphism $f\colon P \to R$ in $\mathbb{E}$ is *Cartesian* if for any $g\colon Q \to R$ in $\mathbb{E}$ with $pg = pf \circ v$ for some $v\colon pQ \to pP$, there exists a unique $h\colon Q \to P$ in $\mathbb{E}$ above $v$ (i.e. $ph = v$) with $f \circ h = g$. The functor $p$ is a *fibration* if for each $R \in \mathbb{E}$ and $u\colon C \to D$ in $\mathbb{C}$ with $pR = D$, there are an object $u^*R$ and a Cartesian morphism $\dot{u}(R)\colon u^*R \to R$ in $\mathbb{E}$. See below.

The category $\mathbb{E}$ is called *total category* and the category $\mathbb{C}$ is called *base category* of the fibration. For an object $C \in \mathbb{C}$, the objects in $\mathbb{E}$ above $C$ forms a category $\mathbb{E}_C$, called *fibre category* above $C$. The fibre category $\mathbb{E}_C$ is the category of "equivalence relations" on $C$.




▶ **Definition 3.3** (fibre category)**.** Let $p\colon \mathbb{E} \to \mathbb{C}$ be a fibration and $C \in \mathbb{C}$. The *fibre category* $\mathbb{E}_C$ over $C$ is the subcategory of $\mathbb{E}$ whose objects are defined by $\mathrm{ob}(\mathbb{E}_C) = \{R \in \mathbb{E} \mid pR = C\}$, and morphisms are defined by $\mathbb{E}_C(Q,R) = \{f \in \mathbb{E}(Q,R) \mid pf = \mathrm{id}_C\}$ for $Q, R \in \mathrm{ob}(\mathbb{E}_C)$.

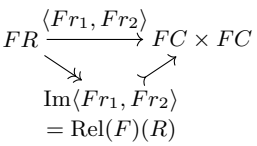
▶ **Example 3.4** (**EqRel** $\to$ **Set** is a fibration)**.** Let **EqRel** be the category of equivalence relations. The objects of **EqRel** are pair $(S,R)$ of a set $S$ and an equivalence relation $R$ on $S$. The morphism $f\colon (S,R) \to (S',R')$ in **EqRel** is a function $f\colon S \to S'$ satisfying $(f(x), f(y)) \in R'$ for all $(x,y) \in R$. We sometimes write just $R$ for $(S,R)$ when no confusion arises. The functor $p\colon$ **EqRel** $\to$ **Set** defined by $p(S,R) = S$ is a fibration.

For a morphism $u\colon C \to D$ in the base category of a fibration, the map $u^*\colon \mathrm{ob}(\mathbb{E}_D) \to \mathrm{ob}(\mathbb{E}_C)$ extends to a functor $u^*\colon \mathbb{E}_D \to \mathbb{E}_C$ between fibre categories. We call the functor $u^*\colon \mathbb{E}_D \to \mathbb{E}_C$ an *inverse image* functor.

Given a fibration $\mathbb{E} \to \mathbb{C}$ and an endofunctor $F\colon \mathbb{C} \to \mathbb{C}$ on the base category, if $R \in \mathbb{E}$ is above $C \in \mathbb{C}$, we would like to get an object in $\mathbb{E}$ above $FC$. A lifting of $F$ specifies the choice of an object above $FC$.

▶ **Definition 3.5** (lifting, fibred lifting)**.** Let $p\colon \mathbb{E} \to \mathbb{C}$ be a fibration and $F\colon \mathbb{C} \to \mathbb{C}$ be a functor. A *lifting* of $F$ is a functor $\overline{F}\colon \mathbb{E} \to \mathbb{E}$ with $p \circ \overline{F} = F \circ p$. A pair $(F, \overline{F})$ of functor $F\colon \mathbb{C} \to \mathbb{C}$ and its lifting $\overline{F}\colon \mathbb{E} \to \mathbb{E}$ is *fibred* if $\overline{F}$ preserves Cartesian morphisms.

When $(F\colon \mathbb{C} \to \mathbb{C}, \overline{F}\colon \mathbb{E} \to \mathbb{E})$ is fibred, for $f\colon C \to D$ in $\mathbb{C}$ and $R \in \mathbb{E}_D$, we have $\overline{F}(f^*R) = (Ff)^*(\overline{F}R)$ in $\mathbb{E}_{FC}$. An important example of a lifting is a relation lifting.




▶ **Definition 3.6** (relation lifting [13])**.** Let $F\colon$ **Set** $\to$ **Set** be a weak pullback preserving functor. We define a lifting $\mathrm{Rel}(F)\colon$ **EqRel** $\to$ **EqRel** of $F$ along the fibration $p\colon$ **EqRel** $\to$ **Set**, called the *relation lifting* of $F$, as follows. For an object

$(C, R) \in \mathbf{EqRel}$, there is the inclusion $\langle r_1, r_2 \rangle \colon R \rightarrowtail C \times C$. We define the relation lifting on the object $R$ by $\mathrm{Rel}(F)(R) = \mathrm{Im}\langle Fr_1, Fr_2 \rangle$, where $\mathrm{Im}\langle Fr_1, Fr_2 \rangle$ is the image factorisation. By the assumption that $F$ preserves weak pullbacks, we can show that $\mathrm{Rel}(F)(R)$ is an equivalence relation. $\mathrm{Rel}(F)$ can be extended to a functor.

In this paper, we deal with a restricted class of fibrations, called $\mathbf{CLat}_\sqcap$-fibrations.

▶ **Definition 3.7** ($\mathbf{CLat}_\sqcap$-fibration)**.** A fibration $p \colon \mathbb{E} \to \mathbb{C}$ is a $\mathbf{CLat}_\sqcap$*-fibration* if each fibre $\mathbb{E}_C$ is a complete lattice and each inverse image functor $u^* \colon \mathbb{E}_D \to \mathbb{E}_C$ preserves meets $\sqcap$.

For a $\mathbf{CLat}_\sqcap$-fibration, there always exists the left adjoint $u_* \colon \mathbb{E}_C \to \mathbb{E}_D$ to an inverse image functor $u^*$, as is well-known (cf. Freyd's adjoint functor theorem). The functor $u_*$ is defined by $u_*(P) = \bigsqcap \{R \in \mathbb{E}_D \mid P \sqsubseteq u^*(R)\}$ on objects. We call $u_*$ a *direct image* functor.

▶ **Example 3.8** ($\mathbf{EqRel} \to \mathbf{Set}$ is a $\mathbf{CLat}_\sqcap$-fibration)**.** The functor $p \colon \mathbf{EqRel} \to \mathbf{Set}$ from Example 3.4 is a $\mathbf{CLat}_\sqcap$-fibration. We describe the inverse image functor $f^*$ and the direct image functor $f_*$ for a function $f \colon S \to S'$. For an equivalence relation $R'$ on $S'$, the inverse image $f^*(R')$ is the equivalence relation $\{(x, y) \in S \times S \mid (f(x), f(y)) \in R'\}$ on $S$. For an equivalence relation $R$ on $S$, the direct image $f_*(R)$ is the *equivalence closure* of the relation $\big\{(f(x), f(y)) \in S' \times S' \,\big|\, (x, y) \in R\big\}$.

## 3.2 Coalgebras and Bisimulations

Coalgebras are widely used as a generalisation of state-based systems [13, 23].

▶ **Definition 3.9** ($F$-coalgebra)**.** Let $\mathbb{C}$ be a category and $F \colon \mathbb{C} \to \mathbb{C}$ be an endofunctor. An $F$*-coalgebra* is a pair $(C, c)$ of an object $C \in \mathbb{C}$ and a morphism $c \colon C \to FC$.

For an $F$-coalgebra $c \colon C \to FC$, $F$ specifies the type of the system, the carrier object $C$ represents the "set of states" of the system, and $c$ represents the transitions in the system. When $\mathbb{C} = \mathbf{Set}$, for an $F$-coalgebra $c \colon C \to FC$ and a state $x \in C$, the element $c(x) \in FC$ represents properties (e.g. acceptance) and successors of $x$.

A major benefit of coalgebras is that their theory is *functor-generic*: by changing a functor $F$, the same theory uniformly applies to a vast variety of systems.

▶ **Example 3.10.** We describe some $F$-coalgebras for functors $F$ on $\mathbf{Set}$.

1. For the powerset functor $\mathcal{P}$, a $\mathcal{P}$-coalgebra $c \colon C \to \mathcal{P}C$ is a *Kripke frame*. For a state $x \in C$, $c(x) \in \mathcal{P}C$ is the set of successors of $x$.
2. Let $\Sigma$ be an alphabet and $N_\Sigma = 2 \times (\mathcal{P}-)^\Sigma$. An $N_\Sigma$-coalgebra $c \colon C \to N_\Sigma C$ is a *non-deterministic automaton* (NA). For a state $x \in C$, let $(b, t) = c(x) \in 2 \times (\mathcal{P}C)^\Sigma$. The state $x$ is accepting iff $b = 1$, and there is a transition $x \xrightarrow{a} y$ in the NA iff $y \in t(a)$.
3. The distribution functor $\mathcal{D}$ is defined on a set $X$ to be $\mathcal{D}X = \{d \colon X \to [0, 1] \mid \{x \in X \mid f(x) \neq 0\}$ is finite and $\sum_{x \in X} d(x) = 1\}$. A $\mathcal{D}$-coalgebra $c \colon C \to \mathcal{D}C$ is a *Markov chain*. For a state $x$, $c(x) \in \mathcal{D}C$ is a probabilistic distribution $C \to [0, 1]$, which represents the probabilities of transitions to successor states of $x$.

We are interested in similarity of states of a state-transition system, where we consider two states to be similar if one state can mimic the transitions of the other. *Bisimilarity* by Park [21] and Milner [19] is a notion that captures such behaviour of states. Hermida and Jacobs [8] formulated bisimilarity as a coinductive relation on a coalgebra, using a fibration.

▶ **Definition 3.11** (bisimulations and the bisimilarity)**.** Let $p\colon \mathbb{E} \to \mathbb{C}$ be a $\mathbf{CLat}_\sqcap$-fibration, $F\colon \mathbb{C} \to \mathbb{C}$ be a functor, $c\colon C \to FC$ be an $F$-coalgebra and $\overline{F}$ be a lifting of $F$. An $(F, \overline{F})$-*bisimulation* is a $c^* \circ \overline{F}$-coalgebra in $\mathbb{E}_C$, that is an object $R \in \mathbb{E}_C$ with $R \sqsubseteq c^* \circ \overline{F}(R)$. By the Knaster–Tarski theorem, there exists the greatest $(F, \overline{F})$-bisimulation $\nu(c^* \circ \overline{F})$ with respect to the order of $\mathbb{E}_C$, and it is called the $(F, \overline{F})$-*bisimilarity*.

In the above definition, the choice of $\overline{F}$ determines a notion of bisimulation. The relation lifting $\mathrm{Rel}(F)$ (Def. 3.6) is often used as a lifting of $F$. For all the functors we consider, the bisimilarity wrt. $\mathrm{Rel}(F)$ coincides with the *behavioural equivalence*, another well-known notion of bisimilarity [13, §4.5].

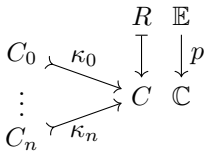
▶ **Example 3.12** ($(F, \mathrm{Rel}(F))$-bisimilarities)**.** We illustrate $(F, \mathrm{Rel}(F))$-bisimilarity (also called *logical $F$-bisimilarity* [13]) for $F$ in Example 3.10. Let $C \in \mathbf{Set}$ and $R \in \mathbf{EqRel}_C$.

1. $(F = \mathcal{P})$. The $(\mathcal{P}, \mathrm{Rel}(\mathcal{P}))$-bisimilarity $\nu(c^* \circ \mathrm{Rel}(\mathcal{P}))$ for a $\mathcal{P}$-coalgebra $c\colon C \to \mathcal{P}C$ is the maximum relation $R$ on $C$ such that if $(x, y) \in R$ then
   - for every $x' \in c(x)$, there is $y' \in c(y)$ such that $(x', y') \in R$, and
   - for every $y' \in c(y)$, there is $x' \in c(x)$ such that $(x', y') \in R$.
2. $(F = N_\Sigma)$. The $(N_\Sigma, \mathrm{Rel}(N_\Sigma))$-bisimilarity $\nu(c^* \circ \mathrm{Rel}(N_\Sigma))$ for an $N_\Sigma$-coalgebra $c\colon C \to N_\Sigma C$ is the ordinary bisimilarity for the NA $c$, that is the maximum relation $R$ on $C$ such that if $(x, y) \in R$ then $\pi_1(c(x)) = \pi_1(c(y))$ and
   - for each $a \in \Sigma, x' \in \pi_2(c(x))(a)$, there is $y' \in \pi_2(c(y))(a)$ such that $(x', y') \in R$, and
   - for each $a \in \Sigma$ and $y' \in \pi_2(c(y))(a)$, there is $x' \in \pi_2(c(x))(a)$ such that $(x', y') \in R$.
3. $(F = \mathcal{D})$ [3]. The $(\mathcal{D}, \mathrm{Rel}(\mathcal{D}))$-bisimilarity $\nu(c^* \circ \mathrm{Rel}(\mathcal{D}))$ for a $\mathcal{D}$-coalgebra $c\colon C \to \mathcal{D}C$ is the maximum relation $R$ such that if $(x, y) \in R$ then $\sum_{z \in K} c(x)(z) = \sum_{z \in K} c(y)(z)$ for every equivalence class $K \subseteq C$ of $R$.

## 4 Fibrational Partitioning

We introduce the notion of fibrational partitioning, one that is central to our algorithm that grows a tree using fibre objects (cf. Fig. 1).

Given an "equivalence relation" $R$ over $C$—identified with an object $R \in \mathbb{E}_C$ over $C \in \mathbb{C}$ in a suitable fibration $p\colon \mathbb{E} \to \mathbb{C}$—a fibrational $R$-partitioning is a mono-sink, shown on the right, that is subject to certain axioms. The notion allows us to explicate equivalence classes (namely $\{C_i\}_i$) in the abstract fibrational language.




▶ **Definition 4.1** ($R$-partitioning)**.** Let $\mathbb{C}$ be a category with pullbacks and an initial object 0, and $p\colon \mathbb{E} \to \mathbb{C}$ be a $\mathbf{CLat}_\sqcap$-fibration. Let $C \in \mathbb{C}$ and $R \in \mathbb{E}_C$. An *$R$-partitioning* is a mono-sink $\{\kappa_i\colon C_i \rightarrowtail C\}_{i \in I}$ to $C$ that satisfies the following conditions.

1. $\kappa_i^*(R) = \top_{C_i}$ for all $i \in I$,
2. $\bigsqcup_{i \in I} (\kappa_i)_*(\top_{C_i}) = R$, and
3. $C_i \not\cong 0$ and $C_i \cap C_j \cong 0$ for each $i, j \in I$ with $i \neq j$.

We say a $\mathbf{CLat}_\sqcap$-fibration $p$ *admits partitioning* if (1) for each $C \in \mathbb{C}$ and $R \in \mathbb{E}_C$, there is an $R$-partitioning; and moreover, (2) the following *monotonicity* holds: for each $C \in \mathbb{C}$, $R, R' \in \mathbb{E}_C$ s.t. $R' \sqsubseteq R$, and each $R$-partitioning $\{\kappa_i\colon C_i \rightarrowtail C\}_{i \in I}$, we have $\bigsqcup_{i \in I} (\kappa_i)_*(\kappa_i^* R') = R'$.

Cond. **3** asserts that the components $C_i$ are nontrivial and disjoint. Cond. **1** says the partitioning $\{C_i\}_i$ is *not too coarse*—the original equivalence $R$, when restricted to $C_i$, should be trivial (which is false when $C_i$ includes a pair that is not $R$-equivalent). Conversely, Cond. **2** means that $\{C_i\}_i$ is *not too fine*—if it were finer than $R$, then the relation $\bigsqcup_{i \in I} (\kappa_i)_*(\top_{C_i})$ over $C$ would be finer than $R$. See the concrete description of $(\kappa_i)_*$ in Example 3.8.

▶ **Example 4.2** (**EqRel** → **Set** admits partitioning). **EqRel** → **Set** admits partitioning. Indeed, given an equivalence relation $R \in \mathbf{EqRel}_C$ over $C$, the mono-sink $\{\kappa_S \colon S \rightarrowtail C\}_{S \in C/R}$, where $S \in C/R$ is naturally identified with a subset of $C$, is an $R$-partitioning. Cond. **1**–**3** are easily verified following Example 3.8.

An $R$-partitioning is not necessarily unique. This happens when $R \in \mathbf{EqRel}_C$ has singleton equivalence classes. Let $A \subseteq C$ be an arbitrary subset such that each $x \in A$ composes a singleton $R$-equivalence class. Then $\{\kappa'_S \colon S \rightarrowtail C\}_{S \in I}$, where $I = (C/R) \setminus \big\{ \{x\} \,\big|\, x \in A \big\}$, is also an $R$-partitioning. With this mono-sink (that is "narrower" than the original $\{\kappa_S\}_{S \in C/R}$), Cond. 2 is satisfied since the equivalence closure operation included in the direct images $(\kappa_i)_*(\top_{C_i})$ (see Example 3.8) compensates the absence of $x \in A$.

We can easily check that **EqRel** → **Set** satisfies monotonicity.

The fibration **EqRel** → **Set** is our leading example, and unfortunately, the only example that we know admits partitioning. There are many other examples of $\mathbf{CLat}_\sqcap$-fibrations (see [17]), but they fail to admit partitioning, typically due to the failure of Cond. 2 of Def. 4.1. This absence of examples does not harm the value of our fibrational framework: our goal is to explicate categorical essences of partition refinement; and we do not aim at new instances via categorical abstraction (although such are certainly desirable).

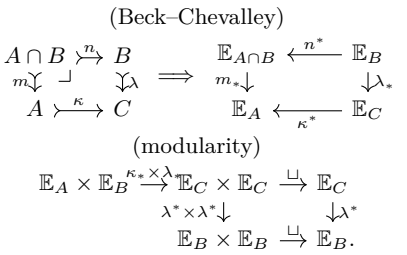



**Figure 3** Conditions for Assum. 4.3.

We introduce further conditions that make fibrations well compatible with partitioning. It is easy to see that **EqRel** → **Set** satisfies the conditions on $p$ in Assum. 4.3.

▶ **Assumption 4.3.** Assume a $\mathbf{CLat}_\sqcap$-fibration $p \colon \mathbb{E} \to \mathbb{C}$ satisfies the following conditions.

1. For each $C \in \mathbb{C}$, the lattice $\mathbf{Sub}(\mathbb{C})_C$ of subobjects of $C$ in $\mathbb{C}$ is distributive.
2. (Beck–Chevalley) For every pullback diagram along monomorphisms in $\mathbb{C}$, shown in the first diagram in Fig. 3, the induced diagram, the second in Fig. 3, commutes.
3. For any monomorphisms $\kappa \colon A \rightarrowtail C$ and $\lambda \colon B \rightarrowtail C$, the third diagram in Fig. 3 is a fork.

## 5 The Naive Fibrational Algorithm $\mathsf{fPR}^{\mathrm{naive}}$

We introduce a naive fibrational partition refinement algorithm, called $\mathsf{fPR}^{\mathrm{naive}}$, as a preparation step to our main algorithm $\mathsf{fPR}^{\mathrm{H}}$ (Algo. 2).

In what follows, a prefix-closed set $T \subseteq \mathbb{N}^*$ (where $\mathbb{N}^*$ is the set of strings over $\mathbb{N}$) is identified with a rooted tree. We denote the leaves of $T$ by $L(T)$.

▶ **Definition 5.1.** Let $p \colon \mathbb{E} \to \mathbb{C}$ be a $\mathbf{CLat}_\sqcap$-fibration that satisfies Assum. 4.3, $F \colon \mathbb{C} \to \mathbb{C}$ be a functor, and $\overline{F} \colon \mathbb{E} \to \mathbb{E}$ be its lifting along $p$ (Def. 3.5). Algo. 1 shows our *naive fibrational partition refinement algorithm*. Given a coalgebra $c \colon C \to FC$, it computes a $\nu(c^*\overline{F})$-partitioning of $C$, i.e. modulo the $(F, \overline{F})$-bisimilarity of $c$ (Def. 3.11).

Algo. 1 starts with $R = \top_C \in \mathbb{E}_C$ and a singleton family of a monomorphism $\{\kappa_\epsilon \colon C_\epsilon \rightarrowtail C\}$. With each iteration, the object $R$ on $C$ gets smaller and closer to $\nu(c^*\overline{F})$ and $R$-partitioning $\{\kappa_\sigma \colon C_\sigma \rightarrowtail C\}_\sigma$ gets finer (see Fig. 4). When the algorithm terminates, $R$ is equal to $\nu(c^*\overline{F})$ and a $\nu(c^*\overline{F})$-partitioning is returned.

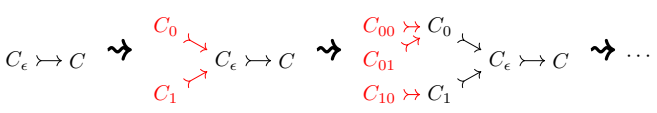



**Figure 4** The $R$-partitioning gets finer as the algorithm runs.

**Algorithm 1** The naive fibrational partition refinement algorithm $\mathsf{fPR}^{\mathrm{naive}}$.

**Input:** A coalgebra $c\colon C \to FC$ in $\mathbb{C}$.
**Output:** A mono-sink $\{\kappa_i\colon C_i \rightarrowtail C\}_{i\in I}$ for some $I$.
1: $T := \{\epsilon\} \subseteq \mathbb{N}^*$; $C_\epsilon := C$; $\kappa_\epsilon := \mathrm{id}_C\colon C_\epsilon \rightarrowtail C$; $R := \top_C$ ▷ initialisation
2: **while** $c^*\overline{F}R \neq R$ **do** ▷ the main loop
3: $R := c^*\overline{F}R$; $L := L(T)$
4: **for** $\sigma \in L$ **do**
5: Take a $\kappa_\sigma^*(R)$-partitioning $\{\lambda_{\sigma,k}\colon C_{\sigma k} \rightarrowtail C_\sigma\}_{k\in\{0,\dots,n_\sigma\}}$ of $C_\sigma$
6: **for** $k = 0, \dots, n_\sigma$ **do** $\kappa_{\sigma k} := \kappa_\sigma \circ \lambda_{\sigma,k}\colon C_{\sigma k} \rightarrowtail C$
7: $T := T \cup \{\sigma 0, \dots, \sigma n_\sigma\}$
8: **return** $\{\kappa_\sigma\colon C_\sigma \rightarrowtail C\}_{\sigma\in L(T)}$

Combining the loop invariant (Lem. 5.2) and termination (Lem. 5.3), we can prove the correctness of the naive algorithm.

▶ **Lemma 5.2** (loop invariant). *At the beginning of each iteration of the main loop, we have*
1. *The mono-sink $\{\kappa_\sigma\colon C_\sigma \rightarrowtail C\}_{\sigma\in L(T)}$ is an $R$-partitioning.*
2. *$\nu(c^*\overline{F}) \sqsubseteq R$.*

▶ **Lemma 5.3** (termination). *If $\mathbb{E}_C$ is a well-founded lattice, then Algo. 1 terminates.*

▶ **Proposition 5.4** (correctness of the naive algorithm). *If $\mathbb{E}_C$ is well-founded, then Algo. 1 terminates and returns $\nu(c^*\overline{F})$-partitioning $\{\kappa\colon C_i \rightarrowtail C\}_{i\in I}$.*

## 6 Optimised Algorithms with Hopcroft's Inequality

Recall that the naive algorithm grows a tree *uniformly* so that every leaf has the same depth (see Fig. 4; note that, even if $C_\sigma$ is fine enough, we extend the node by a trivial partitioning). By selecting leaves in a smart way and generating a tree selectively, the time cost of each iteration can be reduced, so that Hopcroft's inequality is applicable.

In §6.1, we present a functor-generic and fibrational algorithm enhanced with the Hopcroft-type optimisation, calling it $\mathsf{fPR}^{\mathrm{H}}$. We use Hopcroft's inequality (§2) for complexity analysis.

In §6.2 we instantiate $\mathsf{fPR}^{\mathrm{H}}$ to the fibration $\mathbf{EqRel} \to \mathbf{Set}$, obtaining three concrete (yet functor-generic) algorithms $\mathsf{fPR}^{\mathrm{H\text{-}\mathbf{ER}}}_{w_{\mathrm{C}}}$, $\mathsf{fPR}^{\mathrm{H\text{-}\mathbf{ER}}}_{w_{\mathrm{P}}}$, $\mathsf{fPR}^{\mathrm{H\text{-}\mathbf{ER}}}_{w_{\mathrm{R}}}$ that use different weight functions. $\mathsf{fPR}^{\mathrm{H\text{-}\mathbf{ER}}}_{w_{\mathrm{C}}}$ is essentially the algorithm in [14]. The other two ($\mathsf{fPR}^{\mathrm{H\text{-}\mathbf{ER}}}_{w_{\mathrm{P}}}$, $\mathsf{fPR}^{\mathrm{H\text{-}\mathbf{ER}}}_{w_{\mathrm{R}}}$) use the weight functions from the works [6,11,16] on DFA partition refinement. The three algorithms exhibit slightly different asymptotic complexities.

### 6.1 A Fibrational Algorithm $\mathsf{fPR}^{\mathrm{H}}$ Enhanced by Hopcroft's Inequality

We fix a $\mathbf{CLat}_\sqcap$-fibration $p\colon \mathbb{E} \to \mathbb{C}$, functors $F\colon \mathbb{C} \to \mathbb{C}$ and $\overline{F}\colon \mathbb{E} \to \mathbb{E}$, an $F$-coalgebra $c\colon C \to FC$, and a map $w\colon \mathrm{ob}(\mathbf{Sub}(\mathbb{C})_C) \to \mathbb{N}$ (which we use for weights). We write $w(C')$ for $w(\lambda\colon C' \rightarrowtail C)$ when no confusion arises.

The following conditions clarify which properties of $\mathbf{EqRel} \to \mathbf{Set}$ are necessary to make our optimised fibrational algorithm $\mathsf{fPR}^{\mathrm{H}}$ work: the last one (Assum. 6.1.**10**) is for complexity analysis; all the other ones are for correctness of the algorithm.

▶ **Assumption 6.1.** **1.** $\mathbb{C}$ has pullbacks, pushouts along monos, and an initial object 0.
2. The fibre category $\mathbb{E}_0$ above an initial object 0 is trivial, that is $\top_0 = \bot_0$.
3. $\overline{F}$ is a lifting of $F$ along $p$ and $(F, \overline{F})$ is fibred.

**Table 1** (Explicit) correspondence between constructs in $\mathsf{fPR}^{\mathrm{H}}_{(F,\overline{F}),w}$ and the theory in §2

| Constructs in Algo. 2 | Notions in §2 |
|---|---|
| The tree of subobjects $\{C_\sigma\}_{\sigma\in J}$ of $C$, cf. Fig. 5 | A tree $T=(V,E)$, cf. Fig. 2 |
| A set $\{C_\sigma\}_{\sigma\in J}$ of objects in $\mathbb{C}$ | A set $V$ of vertices |
| A set $\{\kappa_{\sigma,k}\colon C_{\sigma k}\rightarrowtail C_\sigma\}_{\sigma k\in J}$ of monomorphisms | A set $E$ of edges |
| A map $w\colon \mathrm{ob}(\mathbf{Sub}(\mathbb{C})_C)\to\mathbb{N}$ with Assum. 6.1.**10** | A weight function $w\colon V\to\mathbb{N}$ (Def. 2.2) |
| A choice of $k_0$ made in Line 10 of Algo. 2 | An hcc $h\colon V\setminus L(T)\to V$ (Def. 2.3) |
| The complexity result of Prop. 6.9 | The inequality of Cor. 2.10 |

4. $F\colon\mathbb{C}\to\mathbb{C}$ preserves monomorphisms whose codomain is not the initial object 0.
5. The fibration $p$ admits monotone partitioning.
6. The fibration $p\colon\mathbb{E}\to\mathbb{C}$ satisfies the three conditions in Assum. 4.3.
7. The fibre category $\mathbb{E}_C$ is a well-founded lattice.
8. If $C'\rightarrowtail C$ and $R\in\mathbb{E}_{C'}$, every $R$-partitioning $\{\lambda_k\colon D_k\rightarrowtail C'\}_{k\in K}$ is finite ($|K|<\infty$).
9. If $\kappa\colon A\rightarrowtail C$ and $\lambda\colon B\rightarrowtail C$ are monomorphisms and $A\cap B\cong 0$, then the functor $\mathbb{E}_A\times\mathbb{E}_B\xrightarrow{\kappa_*\times\lambda_*}\mathbb{E}_C\times\mathbb{E}_C\xrightarrow{\sqcup}\mathbb{E}_C$ is injective on objects.
10. For $C'\rightarrowtail C$, $R\in\mathbb{E}_{C'}$, and $R$-partitioning $\{\kappa_i\colon C_i\rightarrowtail C'\}$ of $C'$, $\sum_{i=1}^n w(C_i)\le w(C')$.

Assum. 6.1.**3** is not overly restrictive. Indeed, the following functors on **Set** have a fibred lifting. The functors described in Example 3.10 are examples of the functor defined by (3).

▶ **Proposition 6.2.** *Consider the class of endofunctors on* **Set** *defined by the BNF below.*

$$F ::= \mathrm{Id} \mid A \mid \coprod_{b\in B} F_b \mid \prod_{b\in B} F_b \mid \mathcal{P}F \mid \mathcal{D}F \qquad \text{where } A \text{ and } B \text{ are sets.} \tag{3}$$

*The relation lifting* $\mathrm{Rel}(F)\colon\mathbf{EqRel}\to\mathbf{EqRel}$ *of* $F$ *(Def. 3.6) is fibred.*

▶ **Proposition 6.3.** *The fibration* $p\colon\mathbf{EqRel}\to\mathbf{Set}$ *with the relation lifting* $\mathrm{Rel}(F)$ *of a functor* $F$ *defined by (3) and a coalgebra* $c\colon C\to FC$ *for a finite set* $C$ *satisfies the assumptions* **1**–**9** *of Assum. 6.1.*

▶ **Definition 6.4** ($\mathsf{fPR}^{\mathrm{H}}$)**.** Let the $\mathbf{CLat}_\sqcap$-fibration $p\colon\mathbb{E}\to\mathbb{C}$, the map $w\colon\mathrm{ob}(\mathbf{Sub}(\mathbb{C})_C)\to\mathbb{N}$, functors $F\colon\mathbb{C}\to\mathbb{C}$ and $\overline{F}\colon\mathbb{E}\to\mathbb{E}$, and the object $C$ satisfy Assum. 6.1. Algo. 2 shows the algorithm $\mathsf{fPR}^{\mathrm{H}}_{(F,\overline{F}),w}$ ($(F,\overline{F})$ and $w$ are omitted when clear from the context). Given a coalgebra $c\colon C\to FC$, it computes a $\nu(c^*\overline{F})$-partitioning of $C$, like the naive algorithm.

The algorithm $\mathsf{fPR}^{\mathrm{H}}$ exposes a tree structure to which Hopcroft's inequality applies. Table 1 summarises how constructs in $\mathsf{fPR}^{\mathrm{H}}$ fit §2. We use Table 1 to explain $\mathsf{fPR}^{\mathrm{H}}$.

Much like $\mathsf{fPR}^{\mathrm{naive}}$ (Algo. 1), $\mathsf{fPR}^{\mathrm{H}}$ grows a tree, as shown in Fig. 5. We take the generated tree as $T=(V,E)$ in §2. Note that, whereas $\mathsf{fPR}^{\mathrm{naive}}$ expands the tree uniformly so that every leaf has the same depth (Fig. 4), $\mathsf{fPR}^{\mathrm{H}}$ expands leaves selectively (Fig. 5).

$\mathsf{fPR}^{\mathrm{H}}$ chooses $k_0$ so that $w(C_{\rho k_0})$ is maximised (Line 10). These choices constitute a heavy child choice (Def. 2.3), an essential construct in Hopcroft's inequality (Thm. 2.9).

$\mathsf{fPR}^{\mathrm{H}}$ starts with $R=\top_C\in\mathbb{E}_C$, the singleton family of a monomorphism $\{\kappa_\epsilon=\mathrm{id}_C\colon C_\epsilon\rightarrowtail C\}$, and a marking $C^{\mathsf{cl}}_\epsilon=0$ of states. For each $\sigma$, $C^{\mathsf{cl}}_\sigma\rightarrowtail C_\sigma$ is a "subset" of $C_\sigma$ consisting of *clean* states; the rest of $C_\sigma$ consists of *dirty* states. Therefore, initially, all states of $C=C_\epsilon$ are marked dirty. The main loop, consisting of **Partitioning** (Line 3–9) and **Relabelling** (Line 10–12), iterates until there is no dirty state (Line 2).

**Algorithm 2** An optimised fibrational partition refinement algorithm $\mathsf{fPR}^{\mathrm{H}}_{(F,\overline{F}),w}$.

**Input:** A coalgebra $c\colon C \to FC$ in $\mathbb{C}$.
**Output:** A mono-sink $\{\kappa_i\colon C_i \rightarrowtail C\}_{i\in I}$ for some $I$.

1: $J := \{\epsilon\} \subseteq \mathbb{N}^*$; $C_\epsilon := C$; $C^{\mathsf{cl}}_\epsilon := 0$ ▷ initialisation
2: **while** there is $\rho \in L(J)$ such that $C^{\mathsf{cl}}_\rho \neq C_\rho$ **do** ▷ the main loop
3: $R := \bigsqcup_{\sigma\in L(J)} (\kappa_\sigma)_*(\top_{C_\sigma})$ ▷ **Partitioning** (Line 3–9)
4: Choose a leaf $\rho \in L(J)$ such that $C^{\mathsf{cl}}_\rho \neq C_\rho$
5: $R_\rho := (c \circ \kappa_\rho)^*(\overline{F}(R))$
6: **if** $R_\rho = \top_{C_\rho}$ **then**
7: $C^{\mathsf{cl}}_\rho := C_\rho$
8: **continue**
9: Take an $R_\rho$-partitioning $\{\kappa_{\rho,k}\colon C_{\rho k} \rightarrowtail C_\rho\}_{k\in\{0,\dots,n_\rho\}}$
10: Choose $k_0 \in \{0,\dots,n_\rho\}$ such that $w(C_{\rho k_0}) = \max_{k\in\{0,\dots,n_\rho\}} w(C_{\rho k})$ ▷ **Relabelling** (Line 10–12)
11: MarkDirty($\rho$, $k_0$)
12: $J := J \cup \{\rho 0,\dots,\rho n_\rho\}$
13: **return** $\{\kappa_\sigma\colon C_\sigma \rightarrowtail C\}_{\sigma\in L(J)}$

14: **procedure** MarkDirty($\rho$, $k_0$)
15: **for** $k \in \{0,\dots,n_\rho\}$ **do** $C^{\mathsf{cl}}_{\rho k} := C_{\rho k}$
16: Let $B$ be the pullback of the diagram:

$$\begin{array}{ccc} B & \rightarrowtail & C \\ \downarrow & \lrcorner & \downarrow{\scriptstyle c} \\ F\left(C_{\rho k_0} \cup \left(\bigcup_{\sigma\in L(J)\setminus\{\rho\}} C_\sigma\right)\right) & \rightarrowtail & FC \end{array}.$$

17: ▷ the bottom morphism is mono by Assum. 6.1.4
18: **for** $\tau \in L(J \cup \{\rho 0,\dots,\rho n_\rho\})$ **do**
19: $C^{\mathsf{cl}}_\tau := C^{\mathsf{cl}}_\tau \cap B$ ▷ states not in $B$ are marked as dirty

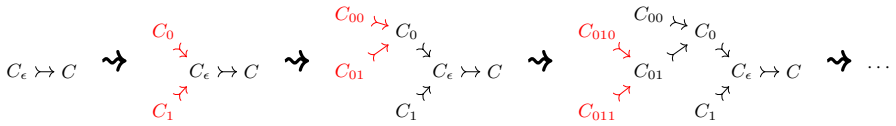



**Figure 5** At each iteration one leaf of the tree is selected and refined.

The **Partitioning** part selects one leaf $C_\rho$ whose states include at least one dirty state (Line 4). The tree is expanded at this selected leaf only. This selection makes Algo. 2 different from Algo. 1, which expands the tree at every leaf (cf. Fig. 4 and Fig. 5).

The **Relabelling** part then updates the clean/dirty marking. Firstly, it chooses one "heavy child" $C_{\rho k_0}$ (Line 10) from the leaves generated in **Partitioning**. Then the iteration calls the MarkDirty procedure (Line 14–19). It first collects states ($B$ in Line 16) whose all "successors" with respect to the coalgebra $c\colon C \to FC$ are in the object $C_{\rho k_0} \cup \left(\bigcup_{\sigma\in L(J)\setminus\{\rho\}} C_\sigma\right)$; the latter intuitively consists of states "unaffected" by tree expansion. The procedure marks only states in $B$ as clean (Line 19), which means that the rest of the states are marked dirty.

Towards the correctness theorem of our optimised fibrational algorithm $\mathsf{fPR}^{\mathrm{H}}$ (Thm. 6.8), we first make a series of preliminary observations. We use the following notations.

▶ **Notation 6.5.** We write $R_i$ for $R$ defined at Line 3 of Algo. 2 at the $i$-th iteration. We write $J_i$ for $J$ at the beginning of the $i$-th iteration. We write $C^{\mathsf{cl},i}_\sigma$ and $\kappa^{\mathsf{cl},i}_\sigma$ for $C^{\mathsf{cl}}_\sigma$ and the monomorphism $\kappa^{\mathsf{cl}}_\sigma\colon C^{\mathsf{cl}}_\sigma \rightarrowtail C$, respectively, at Line 16 at the $i$-th iteration.

We identify loop invariants Prop. 6.6. Termination of $\mathsf{fPR}^{\mathrm{H}}$ follows from Assum. 6.1.7 and 9 (Prop. 6.7). Combining these, we prove the correctness of $\mathsf{fPR}^{\mathrm{H}}$ in Thm. 6.8.

**Algorithm 3** The algorithm $\mathsf{fPR}_w^{\text{H-}\mathbf{ER}}$, obtained as an instance of $\mathsf{fPR}^{\mathrm{H}}$ (Algo. 2) where $p$ is $\mathbf{EqRel} \to \mathbf{Set}$, with a semantically equivalent formulation of MarkDirty (successor-centric in $\mathsf{fPR}^{\mathrm{H}}$; predecessor-centric here in $\mathsf{fPR}_w^{\text{H-}\mathbf{ER}}$).

**Input:** A coalgebra $c\colon C \to FC$ in $\mathbf{Set}$.
**Output:** A mono-sink $\{\kappa_i\colon C_i \rightarrowtail C\}_{i\in I}$ for some $I$.

(the same as Line 1–13 of Algo. 2)

14: **procedure** MarkDirty($\rho, k_0$)
15: **for** $k \in \{0, \dots, n_\rho\}$ **do** $C^{\mathsf{cl}}_{\rho k} := C_{\rho k}$
16: **for** $k \in \{0, \dots, n_\rho\} \setminus \{k_0\}$ and $y \in C_{\rho k}$ **do**
17: **for** $x$: predecessor of $y$ **do**
18: Find $\tau \in L(J \cup \{\rho 0, \dots, \rho n_\rho\})$ such that $x \in C_\tau$
19: If such $\tau$ exists, then $C^{\mathsf{cl}}_\tau := C^{\mathsf{cl}} \setminus \{x\}$ ▷ mark $x$ as dirty

▶ **Proposition 6.6** (loop invariants). *At the beginning of the $i$-th iteration, the following hold.*
1. $(c \circ \kappa_\sigma \circ \kappa_\sigma^{\mathsf{cl},i})^* \overline{F}(R_i) = \top_{C_\sigma^{\mathsf{cl},i}}$ *for each leaf* $\sigma \in L(J_i)$.
2. *The mono-sink* $\{\kappa_\sigma\colon C_\sigma \rightarrowtail C\}_{\sigma\in L(J_i)}$ *is an* $R_i$*-partitioning.*
3. $\nu(c^*\overline{F}) \sqsubseteq R_i$.

*Therefore, after Algo. 2 terminates,* $(c \circ \kappa_\sigma)^* \overline{F} R = \top_{C_\sigma}$ *holds for each* $\sigma \in L(J)$, $\{\kappa_\sigma \rightarrowtail C\}_{\sigma\in L(J)}$ *is an* $R$*-partitioning, and* $\nu(c^*\overline{F}) \sqsubseteq R$, *for* $R \in \mathbb{E}_C$ *defined in Line 3.*

▶ **Proposition 6.7** (termination). *Algo. 2 terminates.*

▶ **Theorem 6.8** (correctness). *Algo. 2 terminates and returns a $\nu(c^*\overline{F})$-partitioning.*

The explicit correspondence between $\mathsf{fPR}^{\mathrm{H}}$ and §2 (Table 1) allows us to directly use Hopcroft's inequality. The following result, while it does not give a complexity bound for $\mathsf{fPR}^{\mathrm{H}}$ itself, plays a central role in the amortised analysis of its concrete instances in §6.2.

▶ **Proposition 6.9.** *If each call of MarkDirty in Algo. 2 takes $\mathcal{O}(K \sum_{k=0}^{n_\rho} w(C_{\rho k}))$ time for some $K$, the total time taken by the repeated calls of MarkDirty is $\mathcal{O}(Kw(C) \log w(C))$.*

## 6.2 Concrete Yet Functor-Generic Algorithms $\mathsf{fPR}_{w_{\mathrm{C}}}^{\text{H-}\mathbf{ER}}$, $\mathsf{fPR}_{w_{\mathrm{P}}}^{\text{H-}\mathbf{ER}}$, $\mathsf{fPR}_{w_{\mathrm{R}}}^{\text{H-}\mathbf{ER}}$

We instantiate the fibrational algorithm $\mathsf{fPR}^{\mathrm{H}}_{(F,\overline{F}),w}$ with $\mathbf{EqRel} \to \mathbf{Set}$ as a base fibration. In this situation, the functor $F$ is an endofunctor on $\mathbf{Set}$ and $\overline{F}$ is an endofunctor on $\mathbf{EqRel}$ which is a fibred lifting of $F$. This instantiation also enables a semantically equivalent reformulation of MarkDirty—its "implementation" is now "predecessor-centric" rather than "successor-centric"—and this aids more refined complexity analysis.

For a weight function $w$ (a parameter of $\mathsf{fPR}^{\mathrm{H}}$), we introduce three examples $w_{\mathrm{C}}, w_{\mathrm{P}}, w_{\mathrm{R}}$, leading to three functor-generic algorithms $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{(F,\overline{F}),w_{\mathrm{C}}}$, $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{(F,\overline{F}),w_{\mathrm{P}}}$ and $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{(F,\overline{F}),w_{\mathrm{R}}}$.

▶ **Definition 6.10** ($\mathsf{fPR}_w^{\text{H-}\mathbf{ER}}$). Let $\mathbf{Set} \xrightarrow{F} \mathbf{Set}$ and $\mathbf{EqRel} \xrightarrow{\overline{F}} \mathbf{EqRel}$ be functors, $C \xrightarrow{c} FC$ be a coalgebra, and $w\colon \mathcal{P}(C) \to \mathbb{N}$ be a function (which amounts to $w\colon \mathrm{ob}(\mathbf{Sub}(\mathbb{C})_C) \to \mathbb{N}$ in §6.1), all satisfying Assum. 6.1 ($C$ must be finite, in particular). The algorithm $\mathsf{fPR}_w^{\text{H-}\mathbf{ER}}$ is shown in Algo. 3; it computes a $\nu(c^*\overline{F})$-partitioning of $C$.

Line 14–19 of Algo. 3 uses this categorical notion of predecessor (Line 17), which is in Def. 6.11. Its equivalence to the original definition (Line 14–19 of Algo. 2) is easy; so $\mathsf{fPR}_w^{\text{H-}\mathbf{ER}}$ is correct by Thm. 6.8. The successor-centric description is more convenient in the correctness proof (Thm. 6.8), while the predecessor-centric one is advantageous for complexity analysis.

**Table 2** Examples of partition refinement algorithms induced by $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{(F,\overline{F}),w}$

| Functor $F(X)$ | Weight function | System | Algorithm |
|---|---|---|---|
| $2 \times X^A$ | $w_{\mathrm{C}}(A)$, $w_{\mathrm{R}}(A)$ | DFA | A variation of Hopcroft's algorithm [6, 11, 16] |
| $\mathcal{P}(A \times X)$ | $w_{\mathrm{C}}(A)$ | LTS | A variation of [24] |
| $\mathcal{D}(X)$ | $w_{\mathrm{C}}(A)$ | Markov chain | A variation of [25] |
| $\mathcal{P}(\mathcal{D}(X))$ | $w_{\mathrm{C}}(A)$ | Markov decision process | A variation of [7] |

▶ **Definition 6.11** (predecessor [14]). Let $c\colon C \to FC$ be a coalgebra in **Set**. For states $x, y \in C$, we say $x$ is a *predecessor* of $y$ if $x \notin B$, where $B$ is a subset of $C$ defined by the following pullback: $\begin{array}{ccc} B & \rightarrowtail & C \\ \downarrow & \lrcorner & \downarrow c \\ F(C \setminus \{y\}) & \xrightarrow[F(i_y)]{} & FC \end{array}$. Here $i_y\colon C \setminus \{y\} \rightarrowtail C$ is the canonical injection.

For a weight function $w$ as a parameter of $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w}$, we introduce three functions.

▶ **Definition 6.12** (weight functions $w_{\mathrm{C}}, w_{\mathrm{P}}, w_{\mathrm{R}}$). We define $w_{\mathrm{C}}, w_{\mathrm{P}}, w_{\mathrm{R}}\colon \mathcal{P}(C) \to \mathbb{N}$—called the *cardinality*, *predecessor* and *reachability* weights, respectively—as follows: $w_{\mathrm{C}}(A) = |A|$, $w_{\mathrm{P}}(A) = \sum_{x \in A} |\{y \in C \mid y \text{ is a predecessor of } x\}|$, and $w_{\mathrm{R}}(A) = |A \cap C'|$, where $C' = \{x \in C \mid x \text{ is a successor of some } y \in C\}$.

The cardinality weight is the most commonly used one in various partition refinement algorithms, including [14]. The latter two have been used in [6, 11, 16] for DFA partition refinement; we use them for the first time in categorical partition refinement algorithms.

Our algorithm $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{(F,\overline{w}),w}$ induces concrete partition refinement algorithms for various systems as shown in Table 2.

The effect of different weight functions is illustrated in Appendix D with an example.

The predecessor-centric MarkDirty and concrete choices of $w$ allow the following fine-grained complexity analysis.

▶ **Theorem 6.13** (complexity of $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{C}}}, \mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{P}}}, \mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{R}}}$).
1. *Let $M = \max_{y \in C} |\{x \in C \mid x \text{ is a predecessor of } y\}|$, the "in-degree" of $c\colon C \to FC$; suppose it takes $\mathcal{O}(f)$ time to compute $c(x) \in FC$ for each $x \in C$. The time complexity of $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{C}}}$ is $\mathcal{O}(fM|C| \log |C|)$.*
2. *Let $m = w_{\mathrm{P}}(C) = \sum_{y \in C} |\{x \in C \mid x \text{ is a predecessor of } y\}|$. The time complexity of $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{P}}}$ is $\mathcal{O}(fm \log m)$. Since $m \leq |C|^2$, it is also bounded by $\mathcal{O}(fm \log |C|)$.*
3. *The time complexity of $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{R}}}$ is $\mathcal{O}(fM|C'| \log |C'|)$, where $f$ and $M$ are from above and $C'$ is from Def. 6.12.*

We note that the complexity bound given in [14] is $\mathcal{O}(fm \log |C|)$ which is the same as **2**: their algorithm is essentially $\mathsf{fPR}^{\text{H-}\mathbf{ER}}_{w_{\mathrm{C}}}$, but their more fine-grained, element-wise analysis derived the aforementioned bound.

We sketch the proof of **1** for illustration. It uses Hopcroft's inequality (Thm. 2.9) in its amortised analysis.

**Proof sketch.** (of Thm. 6.13.**1**) We can check that Algo. 3 (with $w = w_{\mathrm{C}}$) satisfies the premise of Prop. 6.9. By implementing this algorithm properly (preparing a table for $x$ and $\tau$ in Line 18 of Algo. 3, as done in [14]), it takes $\mathcal{O}(1)$ time to execute each iteration of the loop at Line 17. Thus, the loop at Line 16 takes $\mathcal{O}(M|C_{\rho k}|)$ time for each $k$. Therefore the time taken for each call of MarkDirty is $\mathcal{O}(M \sum_{k \in \{0,\dots,n_\rho\} \setminus \{k_0\}} |C_{\rho k}|)$. By Prop. 6.9, the time taken for the repeated calls of MarkDirty in total is $\mathcal{O}(M|C| \log |C|)$.

The complexity of the other parts of the algorithm is also bounded. We write $C_\sigma^{\mathsf{di}}$ for $C_\sigma \setminus C_\sigma^{\mathsf{cl}}$. The computation of $R_\rho$ (Line 3–5 of Algo. 2) takes $\mathcal{O}(f|C_\rho^{\mathsf{di}}|)$, and the computation of $R_\rho$-partitioning (Line 9 of Algo. 2) takes $\mathcal{O}(|C_\rho^{\mathsf{di}}|)$, using appropriate data structures. Hence it takes $\mathcal{O}(f|C_\rho^{\mathsf{di}}|)$ for each iteration of the main loop except for MarkDirty.

Therefore the total time for Algo. 3 except for MarkDirty (let us write $T_{\setminus \textsc{MarkDirty}}$) is $\mathcal{O}(\sum_{\rho \text{ for each iteration}} f|C_\rho^{\mathsf{di}}|)$. We use amortised analysis to bound this sum. Specifically, it is easy to see that the sum $\sum_\rho f|C_\rho^{\mathsf{di}}|$ is bounded by the number of times that states are marked as dirty, multiplied by $f$. Throughout the algorithm, the number of times that states are marked as dirty (at Line 19 of Algo. 3) is at most the time consumed by MarkDirty, which is $\mathcal{O}(M|C|\log|C|)$. Therefore $T_{\setminus \textsc{MarkDirty}}$ is $\mathcal{O}(fM|C|\log|C|)$; so is the total time. ◀

## References


1. Christoph Berkholz, Paul S. Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory Comput. Syst.*, 60(4):581–614, 2017. doi:10.1007/s00224-016-9686-0.
2. Filippo Bonchi, Joshua Holland, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proc. ACM Program. Lang.*, 3(POPL):25:1–25:28, 2019. doi:10.1145/3290338.
3. E.P. de Vink and J.J.M.M. Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach. *Theoretical Computer Science*, 221(1):271–293, 1999. doi:10.1016/S0304-3975(99)00035-3.
4. Hans-Peter Deifel, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Generic partition refinement and weighted tree automata. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods – The Next 30 Years*, pages 280–297, Cham, 2019. Springer International Publishing.
5. Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Efficient coalgebraic partition refinement. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2017.32.
6. David Gries. Describing an algorithm by Hopcroft. *Acta Informatica*, 2(2):97–109, 1973.
7. Jan Friso Groote, Jao Rivera Verduzco, and Erik P. De Vink. An efficient algorithm to determine probabilistic bisimulation. *Algorithms*, 11(9), 2018. doi:10.3390/a11090131.
8. Claudio Hermida and Bart Jacobs. Structural induction and coinduction in a fibrational setting. *Information and Computation*, 145(2):107–152, 1998. doi:10.1006/inco.1998.2725.
9. Johanna Högberg, Andreas Maletti, and Jonathan May. Backward and forward bisimulation minimisation of tree automata. In Jan Holub and Jan Žďárek, editors, *Implementation and Application of Automata*, pages 109–121, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
10. Johanna Högberg, Andreas Maletti, and Jonathan May. Bisimulation minimisation for weighted tree automata. In Tero Harju, Juhani Karhumäki, and Arto Lepistö, editors, *Developments in Language Theory*, pages 229–241, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
11. John E. 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.
12. Bart Jacobs. *Categorical Logic and Type Theory*. Studies in logic and the foundations of mathematics. Elsevier Science, 1999.
13. Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. doi:10.1017/CBO9781316823187.
14. Jules Jacobs and Thorsten Wißmann. Fast coalgebraic bisimilarity minimization. *Proc. ACM Program. Lang.*, 7(POPL), Jan. 2023. doi:10.1145/3571245.

15 Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990. doi:10.1016/0890-5401(90)90025-D.

16 Timo Knuutila. Re-describing an algorithm by Hopcroft. *Theoretical Computer Science*, 250(1):333–363, 2001. doi:10.1016/S0304-3975(99)00150-4.

17 Yuichi Komorida, Shin-ya Katsumata, Nick Hu, Bartek Klin, Samuel Humeau, Clovis Eberhart, and Ichiro Hasuo. Codensity games for bisimilarity. *New Gener. Comput.*, 40(2):403–465, 2022. doi:10.1007/s00354-022-00186-y.

18 Sylvain Lombardy and Jacques Sakarovitch. Morphisms and minimisation of weighted automata. *Fundam. Informaticae*, 186(1-4):195–218, 2022. doi:10.3233/FI-222126.

19 Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., USA, 1989.

20 Edward F. Moore. *Gedanken-Experiments on Sequential Machines*, pages 129–154. Princeton University Press, Princeton, 1956. doi:10.1515/9781400882618-006.

21 David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science*, pages 167–183, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg.

22 Robin Piedeleu, Dimitri Kartsaklis, Bob Coecke, and Mehrnoosh Sadrzadeh. Open system categorical quantum semantics in natural language processing. In Lawrence S. Moss and Pawel Sobocinski, editors, *6th Conference on Algebra and Coalgebra in Computer Science, CALCO 2015, June 24-26, 2015, Nijmegen, The Netherlands*, volume 35 of *LIPIcs*, pages 270–289. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CALCO.2015.270.

23 J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. Modern Algebra. doi:10.1016/S0304-3975(00)00056-6.

24 Antti Valmari. Bisimilarity minimization in $O(m \log n)$ time. In Giuliana Franceschinis and Karsten Wolf, editors, *Applications and Theory of Petri Nets*, pages 123–142, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

25 Antti Valmari and Giuliana Franceschinis. Simple o(m logn) time markov chain lumping. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 38–52, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

26 Thorsten Wißmann, Hans-Peter Deifel, Stefan Milius, and Lutz Schröder. From generic partition refinement to weighted tree automata minimization. *Formal Aspects Comput.*, 33(4-5):695–727, 2021. doi:10.1007/s00165-020-00526-z.

## A Proofs for Section 2 (Hopcroft's Inequality)

▶ **Lemma 2.4.** *Let $T$ be a finite tree with a root $r$, $w$ be a weight function of $T$, and $S$ be an arbitrary set of edges of $T$. Then $\sum_{v\in V(T)} \sum_{\substack{u\in \mathrm{ch}(v)\\(v,u)\notin S}} w(u) \geq \sum_{l\in L(T)} \left|\mathrm{path}(r,l)\setminus S\right| \cdot w(l)$ holds. The equality holds when $w$ is tight.*

**Proof.** Let $r$ be the root of $T$. We prove by the induction on the size of $T$.

(**Base case**). $T$ is a tree whose vertex is only $r$. Thus we have

$$\sum_{v\in V(T)} \sum_{\substack{u\in \mathrm{ch}(v)\\(v,u)\notin S}} w(u) = \sum_{\substack{u\in \varnothing = \mathrm{ch}(r)\\(v,u)\notin S}} w(u) = 0 = \left|\varnothing\right| \cdot w(r) = \sum_{l\in L(T)} \left|\mathrm{path}(r,l)\setminus S\right| \cdot w(l).$$

(**Induction step**). Let $U = \mathrm{ch}(r) \cap S$ and $W = \mathrm{ch}(r) \setminus S$. The following calculation shows the inequality:

$$\begin{aligned}
&\sum_{v\in V(T)} \sum_{\substack{u\in \mathrm{ch}(v)\\(v,u)\notin S}} w(u)\\
&= \sum_{r'\in W} w(r') + \sum_{r'\in U\cup W} \sum_{v\in V(\mathrm{tr}(r'))} \sum_{\substack{u\in \mathrm{ch}(v)\\(v,u)\notin S}} w(u)\\
&\geq \sum_{r'\in W} w(r') + \sum_{r'\in U\cup W} \sum_{l\in L(\mathrm{tr}(r'))} \left|\mathrm{path}(r',l)\setminus S\right| \cdot w(l)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{induction hypothesis}\\
&= \sum_{r'\in U} \sum_{l\in L(\mathrm{tr}(r'))} \left|\mathrm{path}(r',l)\setminus S\right| \cdot w(l) + \sum_{r'\in W} \left( w(r') + \sum_{l\in L(\mathrm{tr}(r'))} \left|\mathrm{path}(r',l)\setminus S\right| \cdot w(l) \right)\\
&\geq \sum_{r'\in U} \sum_{l\in L(\mathrm{tr}(r'))} \left|\mathrm{path}(r',l)\setminus S\right| \cdot w(l) + \sum_{r'\in W} \left( \sum_{l\in L(\mathrm{tr}(r'))} w(l) + \sum_{l\in L(\mathrm{tr}(r'))} \left|\mathrm{path}(r',l)\setminus S\right| \cdot w(l) \right)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by } w(r') \geq \sum_{l\in L(\mathrm{tr}(r'))} w(l)\\
&= \sum_{r'\in U} \sum_{l\in L(\mathrm{tr}(r'))} \left|\mathrm{path}(r',l)\setminus S\right| \cdot w(l) + \sum_{r'\in W} \sum_{l\in L(\mathrm{tr}(r'))} \left(w(l) + \left|\mathrm{path}(r',l)\setminus S\right| \cdot w(l)\right)\\
&= \sum_{r'\in U} \sum_{l\in L(\mathrm{tr}(r'))} \left|\mathrm{path}(r,l)\setminus S\right| \cdot w(l) + \sum_{r'\in W} \sum_{l\in L(\mathrm{tr}(r'))} \left|\mathrm{path}(r,l)\setminus S\right| \cdot w(l)\\
&= \sum_{l\in L(T)} \left|\mathrm{path}(r,l)\setminus S\right| \cdot w(l).
\end{aligned}$$

When $w$ is tight, we have $w(r') = \sum_{l\in L(\mathrm{tr}(r'))} w(l)$ for every $r' \in W$, and the equality holds. ◀

▶ **Lemma 2.8.** *Let $T$ be a finite tree with a root $r$, $w$ be a weight of $T$, and $h$ be an hcc for $w$. For each vertex $v \in V(T)$ with $w(v) \neq 0$, the following inequality holds: $|\mathrm{lpath}(r,v)| \leq \log_2 w(r) - \log_2 w(v)$.*

**Proof.** Let $\langle (v_1,u_1),\dots,(v_m,u_m)\rangle$ be the sequence of the light edges of $\mathrm{lpath}(r,v)$ in the order from $r$ to $v$. Since $u_i \neq h_{v_i}$, we have $2\cdot w(u_i) \leq w(v_i)$. We also have $w(v_{i+1}) \leq w(u_i)$.

Hence, the following inequality holds:

$$\begin{aligned} w(r) &\geq w(v_1) \geq 2 \cdot w(u_1) \\ &\geq 2 \cdot w(v_2) \geq 2^2 \cdot w(u_2) \\ &\dots \\ &\geq 2^{m-1} \cdot w(v_m) \geq 2^m \cdot w(u_m) \\ &\geq 2^m \cdot w(v). \end{aligned}$$

Taking the logarithm of both sides yields $|\operatorname{lpath}(r, v)| = m \leq \log_2 w(r) - \log_2 w(v)$. ◀

▶ **Theorem 2.9** (Hopcroft's inequality)**.** *Let $T$ be a finite tree with root $r$, $w$ be a weight function of $T$, and $h$ be a heavy child choice for $w$. The following inequality holds.*

$$\textstyle\sum_{v \in V(T)} \sum_{u \in \operatorname{lch}_h(v)} w(u) \;\leq\; w(r) \log_2 w(r) - \sum_{l \in L(T), w(l) \neq 0} w(l) \log_2 w(l). \tag{2}$$

**Proof.** Let $w'$ be the tightening of $w$. By Lem. 2.7, $w'$ is a tight weight function, and $h$ is also a heavy child choice for $w'$. If we have

$$\sum_{v \in V(T)} \sum_{u \in \operatorname{lch}_h(v)} w'(u) \;\leq\; w'(r) \log_2 w'(r) - \sum_{l \in L(T), w'(l) \neq 0} w'(l) \log_2 w'(l) \tag{4}$$

then the desired inequality holds:

$$\begin{aligned} \sum_{v \in V(T)} \sum_{u \in \operatorname{lch}_h(v)} w(u) &\leq \sum_{v \in V(T)} \sum_{u \in \operatorname{lch}_h(v)} w'(u) && \text{Lem. 2.7} \\ &\leq w'(r) \log_2 w'(r) - \sum_{\substack{l \in L(T) \\ w'(l) \neq 0}} w'(l) \log_2 w'(l) \\ &\leq w(r) \log_2 w(r) - \sum_{\substack{l \in L(T) \\ w(l) \neq 0}} w(l) \log_2 w(l). && \text{Lem. 2.7} \end{aligned}$$

Now, our goal is to show (4). It is proven by the following calculation:

$$\begin{aligned} &\sum_{v \in V(T)} \sum_{u \in \operatorname{lch}_h(v)} w'(u) \\ &= \sum_{l \in L(T)} \sum_{(v,u) \in \operatorname{lpath}(r,l)} w'(l) && w' \text{: tight, and Lem. 2.5} \\ &= \sum_{\substack{l \in L(T) \\ w'(l) \neq 0}} \sum_{(v,u) \in \operatorname{lpath}(r,l)} w'(l) \\ &\leq \sum_{\substack{l \in L(T) \\ w'(l) \neq 0}} (\log_2 w'(r) - \log_2 w'(l)) \cdot w'(l) && \text{Lem. 2.8} \\ &= \left( \sum_{\substack{l \in L(T) \\ w'(l) \neq 0}} w'(l) \right) \log_2 w'(r) - \sum_{\substack{l \in L(T) \\ w'(l) \neq 0}} w'(l) \log_2 w'(l) \\ &= w'(r) \log_2 w'(r) - \sum_{\substack{l \in L(T) \\ w'(l) \neq 0}} w'(l) \log_2 w'(l). \end{aligned}$$

◀

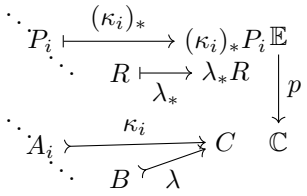
## B Proofs for Section 5 (The Naive Fibrational Algorithm fPR$^{\text{naive}}$)

▶ **Lemma 5.2** (loop invariant). *At the beginning of each iteration of the main loop, we have*
1. *The mono-sink $\{\kappa_\sigma \colon C_\sigma \rightarrowtail C\}_{\sigma \in L(T)}$ is an $R$-partitioning.*
2. *$\nu(c^* \overline{F}) \sqsubseteq R$.*

**Proof.** Firstly we prove the following two lemmas.

▶ **Lemma B.1.** *Assume a $\mathbf{CLat}_\sqcap$-fibration $p \colon \mathbb{E} \to \mathbb{C}$ satisfies Assum. 4.3. Now let $\{\kappa_i \colon A_i \rightarrowtail C\}_{i \in I}$ be a mono-sink in $\mathbb{C}$ that is pairwise disjoint, $\lambda \colon B \rightarrowtail C$ be a monomorphism with $A_i \sqcap B \cong 0$ for each $i \in I$, $P_i \in \mathbb{E}_{A_i}$, and $R \in \mathbb{E}_B$. Then we have $\lambda^* \left( \left( \bigsqcup_{i \in I} (\kappa_i)_* (P_i) \right) \sqcup \lambda_* (R) \right) = R$.*

The above claim can be understood as follows: when a monomorphism $B \rightarrowtail C$ is added to a mono-sink $\{\kappa_i \colon A_i \rightarrowtail C\}_{i \in I}$, if $B$ is disjoint from each $A_i$ ($A_i \sqcap B \cong 0$), then the objects $P_i$ above $A_i$ do not interfere with $R$ above $B$.




Cond. **3** can be equationally expressed by $\lambda^*(\kappa_*(Q) \sqcup \lambda_*(R)) = (\lambda^* \kappa_*(Q)) \sqcup (\lambda^* \lambda_*(R))$; it can be thought of as a fibrational analogue of *modularity* of a lattice.

**Proof.** Let $A = \bigcup_{i \in I} A_i$, $\kappa \colon A \rightarrowtail C$, and $\iota_i \colon A_i \rightarrowtail A$ for each $i \in I$. Notice that the following diagram is a pullback since $\mathbf{Sub}(\mathbb{C})_C$ is distributive (Cond. **1**):

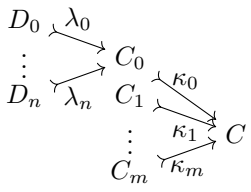
$$\begin{array}{ccc} 0 = B \sqcap \left(\bigcup_{i \in I} A_i\right) & \rightarrowtail & B \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \lambda} \\ \bigcup_{i \in I} A_i & \xrightarrow{\kappa} & C. \end{array} \tag{5}$$

We have

$$\begin{aligned} & \lambda^* \left( \left( \bigsqcup_{i \in I} (\kappa_i)_* (P_i) \right) \sqcup \lambda_* (R) \right) \\ &= \lambda^* \left( \left( \bigsqcup_{i \in I} (\kappa \circ \iota_i)_* (P_i) \right) \sqcup \lambda_* (R) \right) \\ &= \lambda^* \left( \kappa_* \left( \bigsqcup_{i \in I} (\iota_i)_* (P_i) \right) \sqcup \lambda_* (R) \right) && \kappa_* \text{: left adjoint} \\ &= \left( \lambda^* \kappa_* \left( \bigsqcup_{i \in I} (\iota_i)_* (P_i) \right) \right) \sqcup (\lambda^* \lambda_* (R)) && \text{Cond. } \mathbf{3} \\ &= \bot \sqcup R && \text{Equation (5) and Cond. } \mathbf{2} \\ &= R. \end{aligned}$$

◀

The following lemma intuitively says that a partitioning can be refined by another partitioning of one of its leaves, in the manner shown as follows.




▶ **Lemma B.2.** *Let $p\colon \mathbb{E} \to \mathbb{C}$ be a* $\mathbf{CLat}_\sqcap$*-fibration that satisfies the three conditions in Lem. B.1, $C \in \mathbb{C}$, $R \in \mathbb{E}_C$, $\{\kappa_j\colon C_j \rightarrowtail C\}_{j\in\{0,\dots,m\}}$ be an $R$-partitioning of $C$, $R_0$ be an object of $\mathbb{E}_{C_0}$, and $\{\lambda_k\colon D_k \rightarrowtail C_0\}_{k\in\{0,\dots,n\}}$ be an $R_0$-partitioning of $C_0$. The family of monomorphisms*

$$\Pi = \{\kappa_0 \circ \lambda_k\colon D_k \rightarrowtail C\}_{k\in\{0,\dots,n\}} \cup \{\kappa_j\colon C_j \rightarrowtail C\}_{j\in\{1,\dots,m\}}$$

*is a $Q$-partitioning of $C$ where*

$$Q = \left(\bigsqcup_{k=0}^{n} (\kappa_0 \circ \lambda_k)_*(\top_{D_k})\right) \sqcup \left(\bigsqcup_{j=1}^{m} (\kappa_j)_*(\top_{C_j})\right) = (\kappa_0)_*(R_0) \sqcup \left(\bigsqcup_{j=1}^{m} (\kappa_j)_*(\top_{C_j})\right).$$

**Proof.** We check that $\Pi$ satisfies the conditions **1**–**3** of Def. 4.1.

$\Pi$ satisfies Def. 4.1.**2** by the definition of $Q$.

For each $k \in \{0,\dots,n\}$ and $j \in \{1,\dots,m\}$, we have $D_k \cap C_j = 0$. Since $\{\kappa_j\colon C_j \rightarrowtail C\}_{j\in\{0,\dots,m\}}$ and $\{\lambda_k\colon D_k \rightarrowtail C_0\}_{k\in\{0,\dots,n\}}$ are partitioning, we have $C_j \cap C_{j'} = 0$ for $j, j' \in \{1,\dots,m\}$ with $j \neq j'$ and $D_k \cap D_{k'} = 0$ for $k, k' \in \{1,\dots,n\}$ with $k \neq k'$. Moreover, we have $C_j \ncong 0$ and $D_k \ncong 0$ for each $j$ and $k$. Thus, $\Pi$ satisfies Def. 4.1.**3**.

We have

$$\begin{aligned}\kappa_j^*(Q) &= \kappa_j^*\left((\kappa_0)_*(R_0) \sqcup \left(\bigsqcup_{j'=1}^{m} (\kappa_{j'})_*(\top_{C_{j'}})\right)\right)\\ &= \top_{C_j} && \text{Lem. B.1}\end{aligned}$$

for $j \in \{1,\dots,m\}$, and

$$\begin{aligned}&(\kappa_0 \circ \lambda_k)^*(Q)\\ &= (\kappa_0 \circ \lambda_k)^*\left(\left(\bigsqcup_{k'=0}^{n} (\kappa_0 \circ \lambda_{k'})_*(\top_{D_{k'}})\right) \sqcup \left(\bigsqcup_{j=1}^{m} (\kappa_j)_*(\top_{C_j})\right)\right)\\ &= \top_{D_k} && \text{Lem. B.1}\end{aligned}$$

for $k \in \{0,\dots,n\}$. Hence, $\Pi$ satisfies Def. 4.1.**1**. Therefore, $\Pi$ is a $Q$-partitioning of $C$. ◀

We go back to the proof of Lem. 5.2. We write $R_i$ and $T_i$ for $R$ and $T$, respectively, at the beginning (line 2) of the $i$-th iteration of the main loop. We prove **1** and **2** by the induction on $i$.

(**Base case**). We have $R_0 = \top_C$ and $T_0 = \{\epsilon\}$. The mono-sink is $\{\kappa_\epsilon\colon C_\epsilon \rightarrowtail C\}$, and this is an $R_0$-partitioning of $C$. We also have $\nu(c^*\overline{F}) \sqsubseteq \top_C = R_0$.

(**Induction step**). Assume that $\{\kappa_\sigma \colon C_\sigma \rightarrowtail\}_{\sigma \in L(T_i)}$ is an $R_i$-partitioning and $\nu(c^*\overline{F}) \sqsubseteq R_i$. By $\nu(c^*\overline{F}) \sqsubseteq R_i$, we have $\nu(c^*\overline{F}) = c^*\overline{F}(\nu(c^*\overline{F})) \sqsubseteq c^*\overline{F}(R_i) = R_{i+1}$. Hence, **2** holds for $i+1$. By Lem. B.2 and the induction hypothesis, we have $\{\kappa_\sigma \colon C_\sigma \rightarrowtail C\}_{\sigma \in L(T_{i+1})}$ is an $R_{i+1}$-partitioning. Therefore, **1** holds for $i+1$. ◀

▶ **Lemma 5.3** (termination). *If $\mathbb{E}_C$ is a well-founded lattice, then Algo. 1 terminates.*

**Proof.** We write $R_i$ for $R$ at the beginning (line 2) of the $i$-th iteration of the main loop, and have

$$R_0 = \top_C, \quad R_1 = c^*\overline{F}R_0, \quad \dots, \quad R_{i+1} = c^*\overline{F}R_i, \quad \dots.$$

We have $R_1 \sqsubseteq \top_C = R_0$. It is proven by the induction on $i$ and the functoriality of $c^*\overline{F}$ that $R_i = c^*\overline{F}R_{i-1} \sqsubseteq R_{i-1}$ holds for each $i$ such that $R_i$ is defined. By this observation and the condition in line 2, we have the strictly descending sequence in $\mathbb{E}_C$: $R_0 \sqsupset R_1 \sqsupset \cdots \sqsupset R_n \sqsupset \cdots$. Since $\mathbb{E}_C$ is well-founded, the length of the sequence is finite, that is Algo. 1 terminates. ◀

▶ **Proposition 5.4** (correctness of the naive algorithm). *If $\mathbb{E}_C$ is well-founded, then Algo. 1 terminates and returns $\nu(c^*\overline{F})$-partitioning $\{\kappa \colon C_i \rightarrowtail C\}_{i \in I}$.*

**Proof.** By Lem. 5.3, Algo. 1 terminates. From the termination condition (line 2) of the main loop, $R = c^*\overline{F}R$ holds when the algorithm terminates. Hence, $R$ is a fixed point. By Lem. 5.2.**2**, $R$ is greater than or equal to the greatest fixed point $\nu(c^*\overline{F})$ of the functor $c^*\overline{F}$. Thus, we have $R = \nu(c^*\overline{F})$. ◀

## C Proofs for Section 6 (Optimised Algorithms with Hopcroft's Inequality)

▶ **Proposition 6.3.** *The fibration $p \colon \mathbf{EqRel} \to \mathbf{Set}$ with the relation lifting $\mathrm{Rel}(F)$ of a functor $F$ defined by (3) and a coalgebra $c \colon C \to FC$ for a finite set $C$ satisfies the assumptions* **1**–**9** *of Assum. 6.1.*

**Proof.** We only prove that $p \colon \mathbf{EqRel} \to \mathbf{Set}$ satisfies the premise **3** of Lem. B.1. The other conditions are easy to check.

Given injections $\kappa \colon A \rightarrowtail C$ and $\lambda \colon B \rightarrowtail C$ in **Set** and equivalence relations $R$ on $A$ and $S$ on $B$. We want to show $(\lambda^*\kappa_*R) \sqcup (\lambda^*\kappa_*S) = \lambda^*((\kappa_*R) \sqcup (\lambda_*S))$. For any $(x, y) \in \lambda^*((\kappa_*R) \sqcup (\lambda_*S))$, we have $(\lambda(x), \lambda(y)) \in (\kappa_*R) \sqcup (\lambda_*S)$. Hence, there exist $m \in \mathbb{N}$ and $z_0, \dots, z_m \in C$ such that $z_0 = \lambda(x)$, $z_m = \lambda(y)$, and $(z_i, z_{i+1}) \in \kappa_*R$ or $(z_i, z_{i+1}) \in \lambda_*S$ for each $i = 0, \dots, m-1$. We can assume that $z_i \neq z_{i+1}$ and $z_i \in \lambda(B)$ for each $i$ without loss of generality since $\lambda(x), \lambda(y) \in \lambda(B)$. Thus, the sequence $x = \lambda^{-1}(z_0), \lambda^{-1}(z_1), \dots, \lambda^{-1}(z_m) = y$ in $B$ satisfies $(\lambda^{-1}(z_i), \lambda^{-1}(z_{i+1})) \in \lambda^*\kappa_*R$ or $(\lambda^{-1}(z_i), \lambda^{-1}(z_{i+1})) \in \lambda^*\lambda_*S$ for each $i$. This means $(x, y) \in (\lambda^*\kappa_*R) \sqcup (\lambda^*\lambda_*S)$. Hence, $\lambda^*((\kappa_*R) \sqcup (\lambda_*S)) \sqsubseteq (\lambda^*\kappa_*R) \sqcup (\lambda^*\lambda_*S)$ holds. Conversely, we can show $\lambda^*((\kappa_*R) \sqcup (\lambda_*S)) \sqsupseteq (\lambda^*\kappa_*R) \sqcup (\lambda^*\lambda_*S)$ by the similar argument. ◀

▶ **Proposition 6.6** (loop invariants). *At the beginning of the $i$-th iteration, the following hold.*

1. $(c \circ \kappa_\sigma \circ \kappa_\sigma^{\mathsf{cl},i})^*\overline{F}(R_i) = \top_{C_\sigma^{\mathsf{cl},i}}$ *for each leaf $\sigma \in L(J_i)$.*
2. *The mono-sink $\{\kappa_\sigma \colon C_\sigma \rightarrowtail C\}_{\sigma \in L(J_i)}$ is an $R_i$-partitioning.*
3. $\nu(c^*\overline{F}) \sqsubseteq R_i$.

*Therefore, after Algo. 2 terminates, $(c \circ \kappa_\sigma)^*\overline{F}R = \top_{C_\sigma}$ holds for each $\sigma \in L(J)$, $\{\kappa_\sigma \rightarrowtail C\}_{\sigma \in L(J)}$ is an $R$-partitioning, and $\nu(c^*\overline{F}) \sqsubseteq R$, for $R \in \mathbb{E}_C$ defined in Line 3.*

**Proof.** We first prove lemmas to prove the loop invariants. The next technical lemma follows easily from the fibredness of $\overline{F}$.

▶ **Lemma C.1.** *Let $m\colon A \to C$ be a morphism in $\mathbb{C}$ and $P_1, P_2 \in \mathbb{E}_C$. If $m^*P_1 = m^*P_2$, then, for any $C' \in \mathbb{C}$, $\kappa\colon C' \to C$ and $\lambda\colon C' \to FA$ such that $c \circ \kappa = Fm \circ \lambda$, we have $\kappa^*(c^*\overline{F}P_1) = \kappa^*(c^*\overline{F}P_2)$.*

**Proof.** For $j = 1, 2$, we have

$$\begin{aligned} \kappa^*(c^*\overline{F}P_j) &= \lambda^*((Fm)^*\overline{F}P_j) && c \circ \kappa = Fm \circ \lambda \\ &= \lambda^*(\overline{F}(m^*P_j)). && \text{Assum. 6.1.3} \end{aligned}$$

Hence, by the assumption $m^*P_1 = m^*P_2$, we obtain $\kappa^*(c^*\overline{F}P_1) = \lambda^*(\overline{F}(m^*P_1)) = \lambda^*(\overline{F}(m^*P_2)) = \kappa^*(c^*\overline{F}P_2)$. ◀

The next lemma (Lem. C.2) identifies the subobject of $C$ that is "unaffected" by the refinement from $R_i$ to $R_{i+1}$, that is, by the tree expansion at the $i$-th iteration. It is not hard to see that the untouched leaves in $\bigcup_{\sigma \in L(T)\setminus\{\rho\}} C_\sigma$ belong to that "unaffected" part. A crucial observation—central to the Hopcroft-type optimisation—is that *at most one* new child $C_{\rho k_0}$ can also be added, where we pick the heavy one for better complexity.

▶ **Lemma C.2.** *Let $\rho$ be the leaf $\rho \in L(T_i) \setminus L(T_{i+1})$ that was chosen in **Partitioning** of the $i$-th iteration, and $k_0 \in \{0, \dots, n_\rho\}$ be the index that was chosen in **Relabelling** of the $i$-th iteration. We have $m^*(R_i) = m^*(R_{i+1})$ for the morphism $m\colon C_{\rho k_0} \cup \left(\bigcup_{\sigma \in L(T_i)\setminus\{\rho\}} C_\sigma\right) \rightarrowtail C$.*

**Proof.** Let $P = \bigsqcup_{\sigma \in L(T_i)\setminus\{\rho\}} (\kappa_\sigma)_*(\top_{C_\sigma})$, and $R_\rho = \bigsqcup_{k \in \{0,\dots,n_\rho\}} (\kappa'_{\rho k})_*(\top_{C_{\rho k}})$. By Lem. B.1.1, the following diagram is pullback:

$$\begin{array}{ccc} C_{\rho k_0} & \xrightarrow{\kappa'_{k_0}} & C_\rho \\ {\scriptstyle \iota'}\downarrow & \lrcorner & \downarrow{\scriptstyle \kappa_\rho} \\ C_{\rho k_0} \cup \left(\bigcup_{\sigma \in L(T_i)\setminus\{\rho\}} C_\sigma\right) & \xrightarrow{m} & C. \end{array}$$

Thus, the following diagram commutes by Lem. B.1.2.

$$\begin{array}{ccc} \mathbb{E}_{C_{\rho k_0}} & \xleftarrow{(\kappa'_{k_0})^*} & \mathbb{E}_{C_\rho} \\ {\scriptstyle \iota'_*}\downarrow & & \downarrow{\scriptstyle (\kappa_\rho)_*} \\ \mathbb{E}_{C_{\rho k_0} \cup \left(\bigcup_{\sigma \in L(T_i)\setminus\{\rho\}} C_\sigma\right)} & \xleftarrow{m^*} & \mathbb{E}_C \end{array} \tag{6}$$

By chasing the diagram, we have

$$\begin{aligned} m^*((\kappa_\rho)_*(\top_{C_\rho})) &= \iota'_*((\kappa'_{k_0})^*(\top_{C_\rho})) && \text{Equation (6)} \\ &= \iota'_*(\top_{C_{\rho k_0}}), && (\kappa'_{k_0})^*\text{: right adjoint and } \top\text{: limit} \end{aligned} \tag{7}$$

and

$$\begin{aligned} m^*\left((\kappa_\rho)_*(R_\rho)\right) &= \iota'_*\left((\kappa'_{k_0})^*(R_\rho)\right) && \text{Equation (6)} \\ &= \iota'_*(\top_{C_{\rho k_0}}). && \{\kappa'_{\rho k}\colon C_{\rho k} \rightarrowtail C_\rho\}_k \text{ is } R_\rho\text{-partitioning} \end{aligned} \tag{8}$$

There are monomorphisms $\iota_\sigma \colon C_\sigma \rightarrowtail C_{\rho k_0} \cup \bigcup_{\sigma \in L(T_i)\setminus\{\rho\}} C_\sigma$ for each $\sigma \in L(T_i) \setminus \{\rho\}$ with $\kappa_\sigma = m \circ \iota_\sigma$. We have

$$
\begin{aligned}
& m^*(R_i) \\
&= m^* \left( \coprod_{\sigma \in L(T_i)} (\kappa_\sigma)_*(\top_{C_\sigma}) \right) && \text{Definition of } R_i \\
&= m^* \left( (\kappa_\rho)_*(\top_{C_\rho}) \sqcup \coprod_{\sigma \in L(T_i)\setminus\{\rho\}} (\kappa_\sigma)_*(\top_{C_\sigma}) \right) \\
&= m^* \left( (\kappa_\rho)_*(\top_{C_\rho}) \sqcup \coprod_{\sigma \in L(T_i)\setminus\{\rho\}} (m \circ \iota_\sigma)_*(\top_{C_\sigma}) \right) && \kappa_\sigma = m \circ \iota_\sigma \\
&= m^* \left( (\kappa_\rho)_*(\top_{C_\rho}) \sqcup m_* \left( \coprod_{\sigma \in L(T_i)\setminus\{\rho\}} (\iota_\sigma)_*(\top_{C_\sigma}) \right) \right) && m_* \text{: left adjoint} \\
&= m^*(\kappa_\rho)_*(\top_{C_\rho}) \sqcup m^* m_* \left( \coprod_{\sigma \in L(T_i)\setminus\{\rho\}} (\iota_\sigma)_*(\top_{C_\sigma}) \right) && \text{Lem. B.1.3} \\
&= \iota'_*(\top_{C_{\rho k_0}}) \sqcup m^* m_* \left( \coprod_{\sigma \in L(T_i)\setminus\{\rho\}} (\iota_\sigma)_*(\top_{C_\sigma}) \right) && (7)
\end{aligned}
$$

and

$$
\begin{aligned}
& m^*(R_{i+1}) \\
&= m^* \left( \left( \coprod_{k=0}^{n_\rho} (\kappa_\rho \circ \kappa'_{\rho k})_*(\top_{C_{\rho k}}) \right) \sqcup \left( \coprod_{\sigma \in L(T_i)\setminus\{\rho\}} (\kappa_\sigma)_*(\top_{C_\sigma}) \right) \right) && \text{Definition of } R_{i+1} \\
&= m^* \left( (\kappa_\rho)_*(R_\rho) \sqcup \left( \coprod_{\sigma \in L(T_i)\setminus\{\rho\}} (\kappa_\sigma)_*(\top_{C_\sigma}) \right) \right) && (\kappa_\rho)_* \text{: left adjoint} \\
&= m^* \left( (\kappa_\rho)_*(R_\rho) \sqcup \left( \coprod_{\sigma \in L(T_i)\setminus\{\rho\}} (m \circ \iota_\sigma)_*(\top_{C_\sigma}) \right) \right) && \kappa_\sigma = m \circ \iota_\sigma \\
&= m^* \left( (\kappa_\rho)_*(R_\rho) \sqcup m_* \left( \coprod_{\sigma \in L(T_i)} (\iota_\sigma)_*(\top_{C_\sigma}) \right) \right) && m_* \text{: left adjoint} \\
&= m^*(\kappa_\rho)_*(R_\rho) \sqcup m^* m_* \left( \coprod_{\sigma \in L(T_i)} (\iota_\sigma)_*(\top_{C_\sigma}) \right) && \text{Lem. B.1.3} \\
&= \iota'_*(\top_{C_{\rho k_0}}) \sqcup m^* m_* \left( \coprod_{\sigma \in L(T_i)\setminus\{\rho\}} (\iota_\sigma)_*(\top_{C_\sigma}) \right). && (8)
\end{aligned}
$$

Therefore, we obtain $m^*(R_i) = m^*(R_{i+1})$. ◀

We illustrate the proof of Lem. C.2. See Fig. 6. A set $C$ is the disjoint union $C_0 \cup C_1 \cup C_2$, where $C_0 = C_{00} \cup C_{01} \cup C_{02}$. An equivalence relation $R$ on $C$ corresponds to a

partitioning $\{C_0, C_1, C_2\}$, and an equivalence relation $R'$ on $C$ corresponds to a partitioning $\{C_{00}, C_{01}, C_{02}, C_1, C_2\}$. As illustrated in Fig. 6, when restricted to $C_{00} \cup C_1 \cup C_2$, $R$ and $R'$ coincide. This restriction yields the same equivalence relation, and Lem. C.2 formalises this coincidence. An important observation is that, we cannot *add* one extra to the restriction. For example, if we restrict $R$ and $R'$ to $C_{00} \cup C_{01} \cup C_1 \cup C_2$ (with both $C_{00}$ and $C_{01}$ included), we have $m^*(R) \neq m^*(R')$, as depicted in Fig. 7.

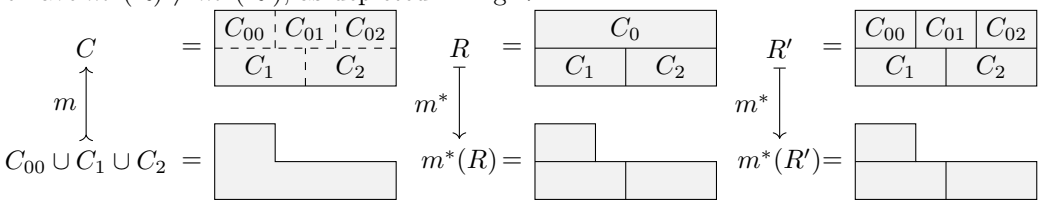



**Figure 6** An example situation of Lem. C.2 in **EqRel** → **Set**: $m^*(R) = m^*(R')$.

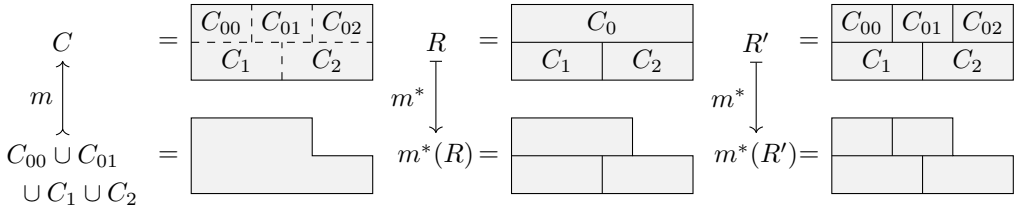
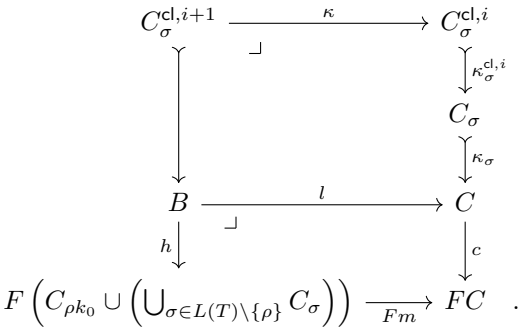



**Figure 7** We cannot add more than one child to the domain of $m$ in Lem. C.2.

We go back to the proof of Prop. 6.6. The claim **2** follows from Lem. B.2 and the induction on $i$. The claim **3** follows from the assumption that $p$ admits monotone partitioning (Assum. 6.1.**5**) and the induction on $i$. We prove **1** by the induction on $i$.

(**Base case**). We have $C_\epsilon^{\mathsf{cl},0} = 0$. Hence, from Assum. 6.1.**2**, we have $(\kappa_\epsilon^{\mathsf{cl},0})^* \kappa_\epsilon^* c^* \overline{F}(R_0) = \top_{C_\epsilon^{\mathsf{cl},0}}$.

(**Induction step**). We assume that **1** holds for $i$, and need to show **1** for $i+1$. For each $\sigma \in L(T_{i+1})$, $C_\sigma^{\mathsf{cl},i+1}$ is defined by the following pullback:




Applying Lem. C.1 to the above diagram and the result of Lem. C.2, we have

$$(\kappa_\sigma^{\mathsf{cl},i+1})^* \kappa_\sigma^* c^* \overline{F} R_{i+1} = (\kappa_\sigma^{\mathsf{cl},i+1})^* \kappa_\sigma^* c^* \overline{F} R_i. \tag{9}$$

If $\sigma \in T_i$, the right-hand side is calculated as:

$$\begin{aligned}
(\kappa_\sigma^{\mathsf{cl},i+1})^* \kappa_\sigma^* c^* \overline{F} R_i &= \kappa^* (\kappa_\sigma^{\mathsf{cl},i})^* \kappa_\sigma^* c^* \overline{F} R_i \\
&= \kappa^* (\top_{C_\sigma^{\mathsf{cl},i}}) && \text{Induction hypothesis} \\
&= \top_{C_\sigma^{\mathsf{cl},i+1}}. && \kappa_* \dashv \kappa^* \text{ and } \top_{C_\sigma^{\mathsf{cl},i}} \text{ is a limit}
\end{aligned}$$

Otherwise, we have $\sigma = \rho k$ for some $k \in \{0, \dots, n_\rho\}$ where $\rho \in L(T_i)$ is the chosen leaf at the **Partitioning** step. Then, we have

$$\begin{aligned}
&(\kappa_{\rho k}^{\mathsf{cl},i+1})^* \kappa_{\rho k}^* c^* \overline{F} R_i \\
&= (\kappa_{\rho k}^{\mathsf{cl},i+1})^* \kappa_{\rho k,k}^* \kappa_\rho^* c^* \overline{F} R_i && \kappa_{\rho k} = \kappa_\rho \circ \kappa_{\rho k,k} \\
&= (\kappa_{\rho k}^{\mathsf{cl},i+1})^* (\top_{C_{\rho k}}) && \text{Def. 4.1.1} \\
&= \top_{C_{\rho k}^{\mathsf{cl},i+1}}. && (\kappa_{\rho k}^{\mathsf{cl},i+1})_* \dashv (\kappa_{\rho k}^{\mathsf{cl},i+1})^* \text{ and } \top_{C_{\rho k}} \text{ is a limit}
\end{aligned}$$

Hence, we have $(\kappa_\sigma^{\mathsf{cl},i+1})^* \kappa_\sigma^* c^* \overline{F} R_{i+1} = \top_{C_\sigma^{\mathsf{cl},i+1}}$ for each $\sigma \in L(T_{i+1})$. ◀

▶ **Proposition 6.7** (termination). *Algo. 2 terminates.*

**Proof.** To prove termination, we use the lexicographical order $\trianglelefteq$ on $\mathbb{E}_C \times \mathbb{N}$, where the order of $\mathbb{N}$ is the usual order $\leq$. The order $\leq$ is well-founded, and $\sqsubseteq$ is well-founded by Assum. 6.1.7. Thus, $\trianglelefteq$ is also well-founded. Let $d_i = |\{\sigma \in L(T_i) \mid C_\sigma^{\mathsf{cl},i} \neq C_\sigma\}|$.

We show that if $C_\rho^{\mathsf{cl},i} \neq C_\rho^i$ holds for some $\rho \in L(T)$, then we have $(R_{i+1}, d_{i+1}) \triangleleft (R_i, d_i)$. Suppose that, at the line 4 of the $i$-th iteration of the main loop, a leaf $\rho \in L(T_i)$ such that $C_\rho^{\mathsf{cl},i} \neq C_\rho^i$ is chosen. We have two cases.

**Case** $(c \circ \kappa_\rho)^*(\overline{F} R_i) \sqsubset \top_{C_\rho}$. In this case, we have $R_i \sqsubset R_{i+1}$ by Assum. 6.1.9. Thus, $(R_{i+1}, d_{i+1}) \triangleleft (R_i, d_i)$ holds.

**Case** $(c \circ \kappa_\rho)^*(\overline{F} R_i) = \top_{C_\rho}$. In this case, the line 7 is executed, and the number of $\sigma \in T$ with $C_\sigma^{\mathsf{cl}} \neq C_\sigma$ reduces: $d_{i+1} < d_i$.

By the above argument shows that, after finite number of iteration, we have $C_\rho^{\mathsf{cl}} = C_\rho$ for all $\rho \in L(T)$. This means that Algo. 2 terminates. ◀

▶ **Theorem 6.8** (correctness). *Algo. 2 terminates and returns a $\nu(c^* \overline{F})$-partitioning.*

**Proof.** Termination is ensured by Prop. 6.7. Let $R$, $T$ and $\{C_\sigma \rightarrowtail C\}_{\sigma \in L(T)}$ be as defined in the last iteration of the main loop. Our goal is to show that $\{C_\sigma \rightarrowtail C\}_{\sigma \in L(T)}$ is a $\nu(c^* \overline{F})$-partitioning of $C$.

We prove $R = c^* \overline{F}(R)$. By Prop. 6.6, $\{C_\sigma \rightarrowtail C\}_{\sigma \in L(T)}$ is an $R$-partitioning. We also have $\kappa_\sigma^*(c^* \overline{F}(R)) = \top_{C_\sigma}$ for each $\sigma \in L(T)$. Thus, we have

$$\begin{aligned}
R &= \bigsqcup_{\sigma \in L(T)} (\kappa_\sigma)_*(\top_{C_\sigma}) && \text{the definition of } R \\
&= \bigsqcup_{\sigma \in L(T)} (\kappa_\sigma)_*(\kappa_\sigma^*(c^* \overline{F}(R))) && \text{Def. 4.1.2} \\
&= c^* \overline{F}(R). && \text{Assum. 6.1.5}
\end{aligned}$$

From Prop. 6.6.3, we have $\nu(c^* \overline{F}) \sqsubseteq R$. Hence, we have $R = \nu(c^* \overline{F})$. ◀

## D Illustration of Runs of Algo. 3

We illustrate Algo. 3 for non-deterministic automata.

▶ **Example D.1.** Let $\Sigma = \{a, b\}$, $C = \{x, y, z, w, v\}$, $N_\Sigma$ be the functor from Example 3.10, and $\overline{N_\Sigma} = \mathrm{Rel}(N_\Sigma)$ (Def. 3.6). We define a coalgebra $c\colon C \to N_\Sigma C$ as shown in Fig. 8. We compare the executions $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma, \overline{N_\Sigma}), w_C}$ and $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma, \overline{N_\Sigma}), w_P}$. At the both initialisations (Line 1), we have a tree whose sole leaf is equal to $C$, which represents the initial partition, and every state in $C$ marked as dirty.

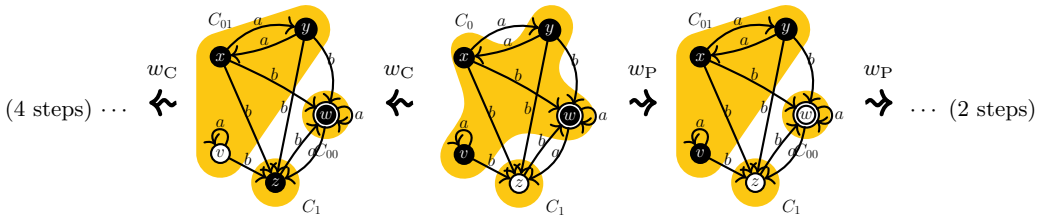



**Figure 8** The snapshots at the end of first and second iterations of $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_C}$ and $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_P}$ for $c$. Yellow areas depict partitions which are refined as the main loop repeats. Clean states are white and dirty states are black.

In the first iteration, both $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_C}$ and $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_P}$ split $C_\epsilon$ and obtain $C_0$ and $C_1$. The $k_0$ chosen at Line 10 is 0 (i.e. $C_0$) in both algorithms, and predecessors of $z$ are marked as dirty (the centre figure of Fig. 8).

In the second iteration, both $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_C}$ and $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_P}$ split $C_0$ and obtain $C_{00}$ and $C_{01}$. The $k_0$ chosen at Line 10 of $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_C}$ is 1 (i.e. $C_{01}$) because $w_C(C_{01}) = 3 > 1 = w_C(C_{00})$, and predecessors of $w$ are marked as dirty (the left figure of Fig. 8).. The $k_0$ chosen at Line 10 of $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_P}$ is 0 (i.e. $C_{00}$) because $w_P(C_{00}) = 4 > 3 = w_P(C_{01})$, and predecessors of the states in $C_{01}$ are marked as dirty (the right figure of Fig. 8).

$\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_P}$ marks states as dirty less than $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_C}$. Hence, $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_P}$ terminates in fewer steps compared with $\mathsf{fPR}^{\text{H-ER}}_{(N_\Sigma,\overline{N_\Sigma}),w_C}$.