Categorical specification and implementation of Replicated Data Types

Fabio Gadducci

Dipartimento di Informatica, Università di Pisa

Hernán Melgratti

ICC – Universidad de Buenos Aires – CONICET, Argentina

Christian Roldán

IMDEA Software Institute

Matteo Sammartino

Royal Holloway University of London, University College London

Abstract

Replicated Data Types (RDTs) have been introduced as an abstraction for dealing with weakly consistent data stores, which may (temporarily) expose multiple, inconsistent views of their state. In the literature, RDTs are usually presented in set-theoretical terms, and only recently different specification flavours have been proposed. This paper offers a categorical presentation for the specification and implementation of RDTs. This paves the way for a method that allows distilling an operational semantics from a specification, which is then exploited to define a notion of implementation correctness via simulation.

Keywords: Replicated data types, Specification, Operational semantics, Functorial characterisation, Implementation correctness

1. Introduction

Modern distributed applications rely on data replication to achieve low latency and high availability even in the presence of failures at the expense of consistency, i.e., applications should tolerate and deal with temporarily inconsistent

[⋆]Research partially supported by the MIUR PRIN 2017FTXR7S "IT-MaTTerS", by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement 778233, by the UBACyT projects 20020170100544BA and 20020170100086BA.

(partial) views of the state. One of the prominent abstractions for programming and reasoning about replicated states are *Replicated Data Types* (RDTs). Roughly, an RDT is an abstract data type whose behaviour embodies replication, i.e., its states describe partial views of the global state and the behaviour of operations is described in terms of the partial views of the state.

Different specification approaches for RDTs have been proposed in the literature [5, 6, 7, 8, 9, 11, 13, 15, 20, 22]. Despite stylistic differences, they abstractly represent the state of a system in terms of two relations defined on executed operations: *visibility*, which explains the partial view of the state over which each operation is executed, and *arbitration*, which totally orders operations and is used for resolving conflicting effects of concurrent operations. Leveraging this abstract view, in this paper we move from a set-theoretic presentation of RDTs to a categorical one, which offers a unifying perspective on the specification and implementation of RDTs. This allows distilling an operational semantics from a specification via a standard categorical construction and, ultimately, leads to a notion of implementation correctness defined as a standard simulation relation.

We start off with an illustrative example of RDTs, and then proceed to outline the contributions of this paper.

1.1. Illustrative example

Consider an RDT *Register* that represents a memory cell whose contents can be updated and read. Following the approach in [11], the RDT *Register* is specified by a function that maps visibility relations into sets of arbitrations: Here we call such function $\mathcal{S}_{lwwR}$. Figure 1a illustrates the definition of $\mathcal{S}_{lwwR}$ for the case in which the visibility relation involves two concurrent writes and a read. Events are depicted by their labels $\langle \texttt{operation}, \mathit{result}\rangle$ where $\texttt{wr}(k)$ stands for an operation that writes the value k and `rd` stands for a read. The two writes are unrelated (i.e., they are not visible to each other), while the read operation sees both writes. The return value of the read operation is `2`, which coincides with one of the visible written values.

According to fig. 1a, $\mathcal{S}_{lwwR}$ maps the visibility graph into a set containing those arbitrations (i.e., total orders over the three events in the visibility relation) in which $\texttt{wr}(1)$ precedes $\texttt{wr}(2)$. Arbitrations may not reflect the causal ordering of events; in fact, the last two arbitrations in the right-hand-side of the equation in Fig. 1a place the operation that read the value 2 before the operation that writes that same value.

We remark that arbitrations do not necessarily account for real-time orderings of events: They are instead possible ways in which events can be *logically* ordered to explain a given visibility. For instance, the excluded arbitrations in the image of $\mathcal{S}_{lwwR}$ are the total orders in which $\texttt{wr}(2)$ precedes $\texttt{wr}(1)$, i.e., the specification bans the behaviour in which a read operation returns a value that is different from the last written one. An extreme situation is the case in which the specification maps a visibility relation into an empty set of arbitrations, which means that events cannot be logically ordered to explain such visibility. For instance, the equation in Fig. 1b assigns an empty set of arbitrations to a visibility relation in which the read operation returns a value that is different

$$\mathcal{S}_{lwwR}\left(\begin{tikzcd} \langle \texttt{wr}(1), ok\rangle \arrow[dr] & & \langle \texttt{wr}(2), ok\rangle \arrow[dl] \\ & \langle \texttt{rd}, 2\rangle & \end{tikzcd}\right) = \left\{ \begin{tikzcd} \langle \texttt{wr}(1), ok\rangle \arrow[d, dash] & \langle \texttt{wr}(1), ok\rangle \arrow[d, dash] & \langle \texttt{rd}, 2\rangle\} \arrow[d, dash] \\ \langle \texttt{wr}(2), ok\rangle \arrow[d, dash] \text{ ,} & \langle \texttt{rd}, 2\rangle\} \arrow[d, dash] \text{ ,} & \langle \texttt{wr}(1), ok\rangle \arrow[d, dash] \\ \langle \texttt{rd}, 2\rangle\} & \langle \texttt{wr}(2), ok\rangle & \langle \texttt{wr}(2), ok\rangle \end{tikzcd} \right\}$$

(a) Visibility relation with admissible arbitrations

$$\mathcal{S}_{lwwR}\left(\begin{tikzcd} \langle \texttt{wr}(1), ok\rangle \arrow[d] \\ \langle \texttt{rd}, 0\rangle \end{tikzcd}\right) = \emptyset$$

(b) Non admissible arbitrations

Figure 1: A register specification

from the unique visible written value (i.e., it returns 0 instead of 1). In this way, the specification bans the behaviour in which a read operation returns a value that does not match a previously written value. As shown in [11], this style of specification can be considered equivalent to (and it is actually more general than) the model for the operational description of RDTs proposed in [7]. We refer the reader to [9] for a formal comparison of the two different approaches.

1.2. Outline of the paper

We now outline the structure of the paper and its main contributions.

Section 2. We start by recalling functional specifications of RDTs, and we provide several examples. We then restrict our attention to *coherent* specifications, which satisfy a suitable condition on arbitrations. We recall two set-theoretic operational models for coherent specifications: the *one-replica Labelled Transition System (LTS)*, modelling the behaviour of a single replica, and the *multi-replica LTS*, modelling the behaviour of multiple replica evolving concurrently.

Section 3. In this section we present our categorical model of coherent specifications. Given a set of labels $\mathcal{L}$ for RDT operations, we model visibilities as $\mathcal{L}$-labelled direct acyclic graphs, and we define a suitable category $\mathbf{PIDag}(\mathcal{L})$ for these graphs. Similarly, we form a category $\mathbf{SPath}(\mathcal{L})$, where objects are sets of $\mathcal{L}$-labelled paths (i.e., total orders) modelling arbitrations. As we shall see, particular care is required in defining morphisms for these categories. Our formulation ensures that certain limits and colimits exist, and that they capture important operations on arbitrations and visibilities. Once the basic categories are in place, we are able to generalise coherent specifications to a class of functors $\mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$, dubbed *coherent functors*. The main result of this section is a sound (Theorem 3.32) and complete (Theorem 3.33) functorial characterisation of coherent specifications, which implies a one-to-one correspondence (Theorem 3.40) between coherent functors and specifications. Furthermore, we show that these results seamlessly extend to specific classes of specifications, namely *saturated* and *topological* ones.

Section 4. RDTs are usually implemented by several replicas that keep their own local state and propagate changes asynchronously. In this section we show that several well-known implementation of common RDTs can be defined in terms of monoidal LTSs, that is, LTSs whose states form a monoid and whose transition relation is compositional with respect to the state combination. The advantage of our formulation is a notion of bisimulation for RDT implementations that is closer to a standard high-order bisimulation (see, e.g., [16]).

Section 5. In this section we exploit the structure of coherent functors to provide a systematic way of recovering an operational semantics from a specification. This is achieved via a standard categorical construction, the so-called *category of elements* of the functor associated with a specification. In particular, we recover both the one-replica (Proposition 5.3) and the multi-replica LTSs (Proposition 5.6). Furthermore, we recover a *contextual* LTS, where labels provide the contextual information that explains how a local computation is embedded into a global context. Implementation correctness is thus straightforwardly stated in terms of a simulation between such LTSs and the one distilled from the specification itself. This characterisation allows for reframing previous ad-hoc formulations of implementation correctness by applying well-known notions from concurrency theory, thus paving the way for the application of more standard techniques in the analysis of RDTs.

Section 6. Finally, in this section we obtain a functorial characterisation of implementations. Implementation functors are defined as functors from a category $\mathbf{IR}(\mathcal{R})$, where arrows represent sequences of actions of a set $\mathcal{R}$ of replica, to a category $\mathbf{P}(\mathbf{Mon})$ representing non-deterministic computations of those replica. The set of states is modelled in the latter category as a monoid, which provides us with a one-to-one representation of the monoidal LTSs for implementations (Theorem 6.6). The category of elements construction again allows us to derive a suitable implementation LTS from implementation functors.

Differences from conference paper. This paper is an extended and revised version of the conference papers [12, 10]. Besides providing clarifications, proofs, and additional examples, in this presentation we give an alternative presentation for implementations, and consequently of their functorial characterisation. Technically, this is achieved by handling operations that mutate the state of an RDT from those that query it differently. Such *mutators* usually preserve the monoidal structure of states, hence the behaviour of a replica (as far as such mutators are concerned) can be described in terms of an LTS that preserves the monoidal structure of the states. This observation allows us to opt for a simpler functorial characterisation of computations in terms of power-monoids instead of the analogous power-domain construction over pre-ordered monoids introduced in [10]. The behaviour of *queries* is captured instead as barbs, i.e., as predicates accounting for the possible observations of states. Implementation equivalence translates to a barbed (higher-order) bisimulation [16], and thus also implementation correctness now relies on barbs.

2. Preliminaries on RDTs

In this section we briefly recall the functional model for RDTs specification and implementation introduced in [11, 9].

2.1. Notation

Let $\mathcal{L}$ be a finite set of *labels*. A *labelled graph* is a triple $\langle \mathcal{E}, \prec, \lambda \rangle$, where $\mathcal{E}$ is the set of vertices (they actually stand for "events", hence the notation), $\prec \subseteq \mathcal{E} \times \mathcal{E}$ is the (directed) connectivity relation, i.e., $\mathtt{e} \prec \mathtt{e}'$ means that there is an edge from $\mathtt{e}$ to $\mathtt{e}'$, and $\lambda \colon \mathcal{E} \to \mathcal{L}$ is a labelling function, assigning a label to each vertex. A graph is *acyclic* if the transitive closure of $\prec$ is a strict partial order. We write $\mathcal{E}_{\mathtt{G}}$, $\prec_{\mathtt{G}}$ and $\lambda_{\mathtt{G}}$ for the corresponding component of a specific graph $\mathtt{G}$. Given two labelled graphs $\mathtt{G}$ and $\mathtt{G}'$, an isomorphism $\varphi \colon \mathtt{G} \cong \mathtt{G}'$ between them is a bijective function $\varphi \colon \mathcal{E}_{\mathtt{G}} \to \mathcal{E}_{\mathtt{G}'}$ that preserves graph structure and labels, i.e., such that $\mathtt{e} \prec_{\mathtt{G}} \mathtt{e}'$ implies $\varphi(\mathtt{e}) \prec_{\mathtt{G}'} \varphi(\mathtt{e}')$, and $\lambda_{\mathtt{G}'}(\varphi(\mathtt{e})) = \lambda_{\mathtt{G}}(\mathtt{e})$, for all $\mathtt{e}, \mathtt{e}' \in \mathcal{E}_{\mathtt{G}}$. A *path* $\langle \mathcal{E}, \leq, \lambda \rangle$ is a graph where $\leq$ is a total order.

Given a graph $\mathtt{G} = \langle \mathcal{E}, \prec, \lambda \rangle$ and a subset $\mathcal{E}' \subseteq \mathcal{E}$, we denote by $\mathtt{G}|_{\mathcal{E}'}$ the obvious restriction (and the same applies to a path $\mathtt{P}$), noting that $\mathtt{G}|_{\emptyset} = \epsilon$, with ϵ the empty graph. Given $\ell \in \mathcal{L}$ and a fresh event $\top$, we denote by $\mathtt{G}^{\ell}_{\mathcal{E}'} = \langle \mathcal{E}_{\top}, \prec \cup (\mathcal{E}' \times \{\top\}), \lambda[\top \to \ell] \rangle$ the *extension* of $\mathtt{G}$ over $\mathcal{E}'$ with ℓ, writing $\mathtt{G}^{\ell}$ when $\mathcal{E}' = \mathcal{E}$.

We denote with $\mathbb{G}(\mathcal{L})$ and $\mathbb{P}(\mathcal{L})$ the collections of (finite) graphs and (finite) paths, respectively, labelled by $\mathcal{L}$. Also, when the set of labels $\mathcal{L}$ is chosen, we let $\mathbb{G}(\mathcal{E}, \lambda)$ and $\mathbb{P}(\mathcal{E}, \lambda)$ be the collections of graphs and paths, respectively, whose set of vertices is $\mathcal{E}$ and which are labelled by $\lambda : \mathcal{E} \to \mathcal{L}$.

2.2. Functional specification of replicated data types

We first recall the notion of (functional) specification. Hereafter we fix a set of labels $\mathcal{L}$.

Definition 2.1 (Specifications)**.** *A* specification $\mathcal{S}$ *is an isomorphism-preserving function* $\mathcal{S} : \mathbb{G}(\mathcal{L}) \to 2^{\mathbb{P}(\mathcal{L})}$ *such that* $\mathcal{S}(\epsilon) = \{\epsilon\}$ *and* $\forall \mathtt{G}.\ \mathcal{S}(\mathtt{G}) \in 2^{\mathbb{P}(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}})}$.

A specification $\mathcal{S}$ maps a graph (interpreted as the visibility relation of a RDT) to a set of paths (that is, the admissible arbitrations of the events). Indeed, each $\mathtt{P} \in \mathcal{S}(\mathtt{G})$ is a path over $\mathcal{E}_{\mathtt{G}}$, hence a total order of the events in $\mathtt{G}$. In specifications we only care about the relative order of events and their labels, hence we require that isomorphic graphs are mapped to sets of paths that are element-wise isomorphic. More precisely, any isomorphism $\varphi \colon \mathtt{G} \cong \mathtt{G}'$ of labelled graphs lifts to the corresponding sets of paths, meaning that $\mathtt{P}' \in \mathcal{S}(\mathtt{G}')$ if and only if there exists $\mathtt{P} \in \mathcal{S}(\mathtt{G})$ such that $\varphi \colon \mathtt{P} \cong \mathtt{P}'$, for all $\mathtt{P}' \in \mathcal{S}(\mathtt{G}')$.

We illustrate the approach by specifying a few well-known examples of RDTs.

2.2.1. Counters

A data type Counter maintains an integer value and provides operations for retrieving and modifying it. We consider the following two variants: increment-only counters (*GCounter*) and positive/negative counters (*PNCounter*). The former account for non-negative counters, which provide the operation `inc` for increments, and the query operation `rd` for retrieving the current value of the counter. The latter may instead contain negative values and provide also the operation `dec` for decrementing. Hereafter, we assume that every counter is initialised in 0. Their specification is given below.

Example 2.2 (*GCounter*)**.** *The specification of the* RDT GCounter *relies on the following set of labels*

$$\mathcal{L}_{\mathcal{S}_{GC}} = \{\langle \texttt{inc}, \mathit{ok}\rangle\} \cup (\{\texttt{rd}\} \times \mathbb{N})$$

Essentially, a label stands for an executed operation (identified by the first component of the pair) and its observed result (given as the second component of the pair). By the definition of $\mathcal{L}_{\mathcal{S}_{GC}}$ above, the execution of operation `inc` *always returns ok (i.e., it succeeds), while* `rd` *returns a natural number. The specification of* GCounter *is given by the function $\mathcal{S}_{GC}$ defined as follows: given a visibility graph* $\mathtt{G}$ *and* $\mathtt{P} \in \mathbb{P}(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}})$

$$\mathtt{P} \in \mathcal{S}_{GC}(\mathtt{G}) \ \textit{iff}\ \forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}}.\ \lambda(\mathtt{e}) = \langle \texttt{rd}, k\rangle \ \Rightarrow\ k = \#P$$

where $P = \{\mathtt{e}' \mid \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e} \wedge \lambda(\mathtt{e}') = \langle \texttt{inc}, \mathit{ok}\rangle\}$. Hereafter, when defining specifications, we will keep the assumption $\mathtt{P} \in \mathbb{P}(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}})$ implicit.

A visibility graph $\mathtt{G}$ *has admissible arbitrations (i.e., $\mathcal{S}_{GC}(\mathtt{G}) \neq \emptyset$) only if every event* $\mathtt{e}$ *in* $\mathtt{G}$ *whose label corresponds to a read operation with return value k (i.e., $\langle \texttt{rd}, k\rangle$) observes (according to* $\mathtt{G}$*) exactly k increment operations.*

We illustrate two cases for the definition of $\mathcal{S}_{GC}$ in Fig. 2. While the configuration in Fig. 2a has admissible arbitrations, the one in Fig. 2b has not, because the unique event labelled by `rd` *returns* 0 *even if it is preceded by an observed increment. In other words, an execution is not allowed to generate such a visibility graph.*

We remark that $\mathcal{S}_{GC}$ does not impose any constraint on the ordering $<_{\mathtt{P}}$. In fact, a path $\mathtt{P} \in \mathcal{S}_{GC}(\mathtt{G})$ does not need to reflect the ordering of $\mathtt{G}$ *as, for instance, the rightmost path in the set of Fig. 2a.*

Example 2.3 (PNCounter)**.** *A* PNCounter *extends the definition of a* GCounter *by allowing decrement operations and negative values, which is reflected by the considered set of labels*

$$\mathcal{L}_{\mathcal{S}_{PNC}} = \{\langle \texttt{inc}, \mathit{ok}\rangle, \langle \texttt{dec}, \mathit{ok}\rangle\} \cup (\{\texttt{rd}\} \times \mathbb{Z})$$

The corresponding specification $\mathcal{S}_{PNC}(\mathtt{G})$ is defined such that the following holds

$$\mathtt{P} \in \mathcal{S}_{PNC}(\mathtt{G}) \ \textit{iff}\ \begin{cases} \forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}}.\ \lambda(\mathtt{e}) = \langle \texttt{rd}, k\rangle \ \Rightarrow k = P - N \ \textit{where} \\ \qquad P = \#\{\mathtt{e}' \mid \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e} \wedge \lambda(\mathtt{e}') = \langle \texttt{inc}, \mathit{ok}\rangle\} \\ \qquad N = \#\{\mathtt{e}' \mid \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e} \wedge \lambda(\mathtt{e}') = \langle \texttt{dec}, \mathit{ok}\rangle\} \end{cases}$$

$$\mathcal{S}_{GC}\left(\begin{array}{c}\langle \mathtt{inc}, ok\rangle\\ \downarrow\\ \langle \mathtt{rd}, 1\rangle\end{array}\right) = \left\{\begin{array}{cc}\langle \mathtt{inc}, ok\rangle & \langle \mathtt{rd}, 1\rangle\\ | & |\\ \langle \mathtt{rd}, 1\rangle & \langle \mathtt{inc}, ok\rangle\end{array}\right\}$$

(a)

$$\mathcal{S}_{GC}\left(\begin{array}{c}\langle \mathtt{inc}, ok\rangle\\ \downarrow\\ \langle \mathtt{rd}, 0\rangle\end{array}\right) = \emptyset$$

(b)

Figure 2: *Counter* specification.

A graph G *has admissible arbitrations when each read returns a value k that corresponds to the difference between the number of increments and decrements observed by that read.*

2.2.2. Registers

A data type Register maintains a value and provides two operations, one for updating it and another one for retrieving the current value. We provide specification for two alternative semantics, namely, a *Multi-value register* and a *Last-write wins* register.

Example 2.4 (*Multi-value register*)**.** *A common abstraction of a memory cell in a replicated system is given by a* Multi-value register*. Differently from a traditional register, a multi-value one may contain several values when it is updated concurrently. Hence, we can fix the following set of labels*

$$\mathcal{L}_{mvR} = \{\langle \mathtt{wr}(k), ok\rangle \mid k \in \mathcal{V}\} \cup (\{\mathtt{rd}\} \times 2^{\mathcal{V}})$$

where $\langle \mathtt{wr}(k), ok\rangle$ *stands for an operation that writes the integer* k *and* $\langle \mathtt{rd}, S\rangle$ *for a read that retrieves the (possibly empty) set of values* S *stored in the register. The return value of every write operation is ok since they always succeed.*

The specification is given by $\mathcal{S}_{mvR} : \mathbb{G}(\mathcal{L}_{mvR}) \to 2^{\mathbb{P}(\mathcal{L}_{mvR})}$ *defined as follows*

$$\mathtt{P} \in \mathcal{S}_{mvR}(\mathtt{G}) \text{ iff } \left\{\begin{array}{l}\forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}}.\\ \lambda(\mathtt{e}) = \langle \mathtt{rd}, S\rangle \Rightarrow S = \{k \mid \exists \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}.\ \lambda(\mathtt{e}') = \langle \mathtt{wr}(k), ok\rangle \wedge \\ \qquad\qquad (\forall \mathtt{e}'' \prec_{\mathtt{G}} \mathtt{e}, k'.\ \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}'' \Rightarrow \lambda(\mathtt{e}'') \neq \langle \mathtt{wr}(k'), ok\rangle)\}\end{array}\right.$$

The condition on the right requires that any event e *in* G *associated with a read (i.e., labelled by* $\langle \mathtt{rd}, S\rangle$*) returns a set* S *that contains all values written by maximal (according to* $\prec_{\mathtt{G}}$*) concurrent updates seen by it. If this is the case, all arbitrations are admissible, i.e.,* $\mathtt{P} \in \mathcal{S}_{mvR}(\mathtt{G})$ *for all* P*, otherwise* $\mathcal{S}_{mvR}(\mathtt{G}) = \emptyset$*.*

An instance of $\mathcal{S}_{mvR}$ *is shown in Fig. 3, where* G *consists of three writes:* $\mathtt{wr}(2)$ *overwrites* $\mathtt{wr}(1)$ *and both are concurrent with* $\mathtt{wr}(3)$*. Additionally, there*

$$\mathcal{S}_{mvR}\left(\begin{tikzcd}[column sep=small, row sep=small] \langle \mathtt{wr}(1), ok\rangle \arrow[d] \arrow[dd, bend right] \arrow[ddr] & \langle \mathtt{wr}(3), ok\rangle \arrow[dd] \\ \langle \mathtt{wr}(2), ok\rangle \arrow[d] \arrow[dr] & \\ \langle \mathtt{rd}, \{2\}\rangle \arrow[r] & \langle \mathtt{rd}, \{2,3\}\rangle \end{tikzcd}\right) = \left\{\begin{array}{cccc} \langle \mathtt{wr}(1), ok\rangle & \langle \mathtt{wr}(1), ok\rangle & \langle \mathtt{rd}, \{2,3\}\rangle & \cdots \\ \mid & \mid & \mid & \\ \langle \mathtt{wr}(2), ok\rangle & \langle \mathtt{wr}(2), ok\rangle & \langle \mathtt{wr}(3), ok\rangle & \cdots \\ \mid & \mid & \mid & \\ \langle \mathtt{rd}, \{2\}\rangle & \langle \mathtt{rd}, \{2\}\rangle & \langle \mathtt{wr}(1), ok\rangle & \cdots \\ \mid & \mid & \mid & \\ \langle \mathtt{wr}(3), ok\rangle & \langle \mathtt{rd}, \{2,3\}\rangle & \langle \mathtt{wr}(2), ok\rangle & \cdots \\ \mid & \mid & \mid & \\ \langle \mathtt{rd}, \{2,3\}\rangle & \langle \mathtt{wr}(3), ok\rangle & \langle \mathtt{rd}, \{2\}\rangle & \cdots \end{array}\right\}$$

Figure 3: Specification of a *Multi-value register*

are two reads: One observes all writes (right-most at the bottom), the other does not see `wr`(3) *(left-most one). Both reads return the set of values written by the maximal observed events: None of them returns* 1 *because it has been overwritten by* 2. *A graph is mapped by* $\mathcal{S}_{mvR}$ *to* $\emptyset$ *if it describes an inconsistent configuration, e.g., if the return value of one read in Fig. 3 were changed to* $\{1\}$.

According to $\mathcal{S}_{mvR}$, *events can be arbitrated in any order, allowing read events to happen before observed writes (as in the second and third path in Fig. 3). This is a common approach in the specification of* RDT*s [7] because it allows permissive strategies for implementation (we refer to [9] for details). If needed, a specification can exclude arbitrations (as illustrated in Ex. 2.5).*

Example 2.5 (*Last-write wins Register*)**.** *An alternative to the* Multi-value Register *is the* Last-write wins Register, *in which every read returns the last written value according to arbitration. We take the following set of labels*

$$\mathcal{L}_{\mathcal{S}_{lwwR}} = \{\langle \mathtt{wr}(k), ok\rangle, \langle \mathtt{rd}, k\rangle \mid k \in \mathcal{V}\} \cup \{\langle \mathtt{rd}, \bot\rangle\}$$

where $\bot$ *is the initial value of a register. Its specification* $\mathcal{S}_{lwwR}$ *is given by*

$$\mathtt{P} \in \mathcal{S}_{lwwR}(\mathtt{G}) \textit{ iff } \left\{\begin{array}{l} \forall \mathsf{e} \in \mathcal{E}_{\mathtt{G}}. \\ \lambda(\mathsf{e}) = \langle \mathtt{rd}, \bot\rangle \Rightarrow \forall \mathsf{e}' \prec_{\mathtt{G}} \mathsf{e}, k'.\ \lambda(\mathsf{e}') \neq \langle \mathtt{wr}(k'), ok\rangle \wedge \\ \lambda(\mathsf{e}) = \langle \mathtt{rd}, k\rangle \Rightarrow \exists \mathsf{e}' \prec_{\mathtt{G}} \mathsf{e}.\ \lambda(\mathsf{e}') = \langle \mathtt{wr}(k), ok\rangle \wedge \\ \qquad (\forall \mathsf{e}'' \prec_{\mathtt{G}} \mathsf{e}, k'.\ \mathsf{e}' <_{\mathtt{P}} \mathsf{e}'' \Rightarrow \lambda(\mathsf{e}'') \neq \langle \mathtt{wr}(k'), ok\rangle) \end{array}\right.$$

According to $\mathcal{S}_{lwwR}$, *a read returns* $\bot$ *when it does not observe any write. On the contrary, a read* e *returns a natural number* k *when it observes some event* e' *that writes* k. *In such case, the arbitration* $\mathtt{P}$ *must order* e' *as the maximal event (accordingly to* $<_{\mathtt{P}}$*) among all write operations seen by* e. *In this way, the specification constrains the allowed arbitrations of a graph.*

2.3. Sets

A data type for sets provides the operations for adding, removing and examining the elements within the set. Different alternatives have been proposed in the literature for resolving conflicts in the presence of concurrent additions and removals; we illustrate two possible alternatives below.

Example 2.6 (*Observed-remove set*). *An* Observed-remove set *(*Or-Set*) is an abstraction for sets in which conflicting concurrent additions and removals are resolved by making the former to win over the latter. Since it is an abstraction for sets, it provides the operations to add and remove elements, and also to lookup its content. Hence, we take the following set of labels*

$$\mathcal{L}_{Or\text{-}Set} = \{\langle \mathtt{add}(k), ok\rangle, \langle \mathtt{rem}(k), ok\rangle \mid k \in \mathcal{V}\} \cup (\{\mathtt{lookup}\} \times 2^{\mathcal{V}})$$

The specification of the Or-Set *is as follows*

$$\mathtt{P} \in \mathcal{S}_{Or-Set}(\mathtt{G}) \text{ iff } \begin{cases} \forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}}. \\ \lambda(\mathtt{e}) = \langle \mathtt{lookup}, S\rangle \Rightarrow \\ \qquad S = \{k \mid \exists \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}.\ \lambda(\mathtt{e}') = \langle \mathtt{add}(k), ok\rangle \ \wedge \\ \qquad\quad (\forall \mathtt{e}'' \prec_{\mathtt{G}} \mathtt{e}.\ \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}'' \Rightarrow \lambda(\mathtt{e}'') \neq \langle \mathtt{rem}(k), ok\rangle)\} \end{cases}$$

The value S returned by a read event e *contains all those elements that have been added but not subsequently removed according to the visibility of* e.

Example 2.7 (2P-Set). *A* Two-Phase set *(*2P-Set*) is an abstraction in which elements can be added and removed; however, after being removed, an element cannot be re-added to the set. For its specification, we fix a set of labels identical to the OR-Set, namely*

$$\mathcal{L}_{2P\text{-}Set} = \{\langle \mathtt{add}(k), ok\rangle, \langle \mathtt{rem}(k), ok\rangle \mid k \in \mathcal{V}\} \cup (\{\mathtt{lookup}\} \times 2^{\mathcal{V}})$$

Its specification of then defined as follows

$$\mathtt{P} \in \mathcal{S}_{2P-Set}(\mathtt{G}) \text{ iff } \begin{cases} \forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}}. \\ \lambda(\mathtt{e}) = \langle \mathtt{lookup}, S\rangle \Rightarrow \\ \qquad S = \{k \mid \exists \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}.\ \lambda(\mathtt{e}') = \langle \mathtt{add}(k), ok\rangle \ \wedge \\ \qquad\qquad \forall \mathtt{e}'' \prec_{\mathtt{G}} \mathtt{e}. \lambda(\mathtt{e}'') \neq \langle \mathtt{rem}(k), ok\rangle)\} \end{cases}$$

The value S returned by a read event e *contains all those elements that have been added but not removed according to the visibility of* e *(regardless of whether the addition took place before or after the removal).*

2.4. Coherent Specifications

We now restrict our attention to *coherent* specifications, which suffice for the standard specification of RDTs [9] and are amenable to a categorical characterisation, as illustrated in the next section. Coherence expresses that admissible arbitrations of a visibility graph are obtained by composing the admissible arbitrations corresponding to smaller visibilities. Its formal definition relies on an auxiliary operation for composing sets of paths. We say that the paths of a set $\mathcal{X} = \{\mathtt{P}_i\}_{i \in I}$ are *compatible* if we have $\lambda_j(\mathtt{e}) = \lambda_k(\mathtt{e})$ for all $\mathtt{e} \in \mathcal{E}_j \cap \mathcal{E}_k$.

Definition 2.8 (Product). *The product of a finite set $\mathcal{X}$ of compatible paths is*

$$\bigotimes \mathcal{X} = \{\mathtt{P} \mid \mathtt{P} \ \text{ is a path over } \bigcup_i \mathcal{E}_i \text{ and } \mathtt{P}|_{\mathcal{E}_i} \in \mathcal{X}\ \}$$

The product of paths is analogous to the synchronous product of transition systems: Common elements are identified and the remaining ones can be freely interleaved, as long as the original orders are respected. Clearly, the result is $\emptyset$ if so is $\mathcal{X}$, while if the set contains only the empty path ϵ the result is $\{\epsilon\}$.

A set of sets of paths $\mathcal{X}_1, \mathcal{X}_2, \dots$ is compatible if $\bigcup_i \mathcal{X}_i$ is so, and we can define $\bigotimes_i \mathcal{X}_i$ as $\bigotimes \bigcup_i \mathcal{X}_i$. But first, for an element $\mathtt{e}$ of a graph $\mathtt{G}$, we denote by $[\mathtt{e}]$ the set of elements of $\mathtt{G}$ that precede $\mathtt{e}$, namely, $\{\mathtt{e}' \mid \mathtt{e}' \prec^*_{\mathtt{G}} \mathtt{e}\}$.

Definition 2.9 ((Past-)Coherent Specification)**.** *Let $\mathcal{S}$ be a specification. We say that $\mathcal{S}$ is past-coherent (briefly, coherent) if* $\forall \mathtt{G} \neq \epsilon.\ \mathcal{S}(\mathtt{G}) = \bigotimes_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} \mathcal{S}(\mathtt{G}|_{[\mathtt{e}]})$.

In a coherent specification $\mathcal{S}$ the arbitrations of a configuration $\mathtt{G}$ (i.e., the set of paths $\mathcal{S}(\mathtt{G})$) are the composition of the arbitrations of its sub-graphs $\mathtt{G}|_{[\mathtt{e}]}$. It can be shown that the specifications in Section 2.2 are coherent.

Coherent specifications can be given an operational interpretation representing admissible computations. We start by defining a *one-replica Labelled Transition System (LTS)*, modelling the behaviour of a single replica.

Definition 2.10 (One-replica LTS)**.** *Given a specification $\mathcal{S}$, its* one-replica LTS *is defined as follows*

- *states are pairs* $\langle \mathtt{G}, \mathtt{P} \rangle$ *such that* $\mathtt{P} \in \mathcal{S}(\mathtt{G})$*;*
- *transitions are of the form* $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\ell} \langle \mathtt{G}^\ell, \mathtt{P}' \rangle$*, where* $\mathtt{P}'|_{\mathcal{E}_{\mathtt{G}}} = \mathtt{P}$.

The behaviour of a single replica that implements a specification $\mathcal{S}$ can be defined in terms of the evolution of the relations $\mathtt{G}$ and $\mathtt{P}$, which respectively account for the visibility and arbitration of the events known by that particular replica. Only pairs $\langle \mathtt{G}, \mathtt{P} \rangle$ in which $\mathtt{P}$ is an admissible arbitration of $\mathtt{G}$, i.e., such that $\mathtt{P} \in \mathcal{S}(\mathtt{G})$, are considered. Each transition $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\ell} \langle \mathtt{G}^\ell, \mathtt{P}' \rangle$ represents the execution of an operation ℓ over the state $\langle \mathtt{G}, \mathtt{P} \rangle$. Such execution extends the visibility relation $\mathtt{G}$ with a top event labelled by ℓ (i.e., $\mathtt{G}^\ell$), which accounts for the fact that the operation observes all the events known by the replica. Since $\langle \mathtt{G}^\ell, \mathtt{P}' \rangle$ should be a state of the LTS, $\mathtt{P}'$ is a total order of the events in $\mathtt{G}^\ell$. The condition $\mathtt{P}'|_{\mathcal{E}_{\mathtt{G}}} = \mathtt{P}$ ensures that $\mathtt{P}'$ preserves the arbitration $\mathtt{P}$, i.e., $\mathtt{P}'$ is obtained by inserting the fresh event labelled by ℓ in any position within $\mathtt{P}$.

Example 2.11. *Consider the* RDT GCounter *specified in 2.2.1. Suppose we have a replica that is aware of the execution of three increment operations, two of which have been performed locally, and another one has been (concurrently) executed over a different replica. An abstract representation of the state of that replica is given by the visibility relation* $\mathtt{G}$ *depicted below and a total order* $\mathtt{P}$ *over those events (any total order would work in this case because arbitration is uninfluential for* GCounter*).*

$\mathtt{G}$

$$\begin{tikzcd}
\langle \mathtt{inc}, ok\rangle \arrow[d] & \langle \mathtt{inc}, ok\rangle \\
\langle \mathtt{inc}, ok\rangle &
\end{tikzcd}$$

Note that $\mathtt{G}$ *abstracts away from the identity of the replicas over which the operations have been executed and only keeps track of their (visibility) dependencies: Each pair of unrelated events corresponds to operations concurrently executed over different replicas. A state of a single replica is associated with a visibility graph accounting for operations executed over different replicas; in this way, we abstractly represent the propagation of events that occurs in concrete implementations. In this scenario, the pair* $\langle \mathtt{G}, \mathtt{P}\rangle$ *can be regarded as the state of a replica that has locally performed two increments and then has received an increment propagated from other replica. Interestingly,* $\langle \mathtt{G}, \mathtt{P}\rangle$ *also describes several other possibilities; e.g., a replica that has performed one increment and then has received information about two (sequential) increments from a different replica; or a third replica that has received the updates propagated by other two replicas. In any case, the behaviour of the replica is just determined by* $\langle \mathtt{G}, \mathtt{P}\rangle$*. The transitions from* $\langle \mathtt{G}, \mathtt{P}\rangle$ *have the following shape* $\langle \mathtt{G}, \mathtt{P}\rangle \xrightarrow{\ell} \langle \mathtt{G}^{\ell}, \mathtt{P}'\rangle$ *where* $\ell \in \mathcal{L}_{\mathcal{S}_{GC}} = \{\langle \mathtt{inc}, ok\rangle\} \cup (\{\mathtt{rd}\} \times \mathbb{N})$ *and* $\mathtt{G}^{\ell}$ *is defined as follows*

$\mathtt{G}^{\ell}$

$$\begin{tikzcd}
\langle \mathtt{inc}, ok\rangle \arrow[d] \arrow[ddr] & \langle \mathtt{inc}, ok\rangle \arrow[dd] \\
\langle \mathtt{inc}, ok\rangle \arrow[dr] & \\
& \ell
\end{tikzcd}$$

If $\ell = \langle \mathtt{inc}, ok\rangle$*, then it suffices to take* $\mathtt{P}'$ *as one of the possible extensions of* $\mathtt{P}$ *(where the new event can be added in any position in* $\mathtt{P}$*). If* $\ell = \langle \mathtt{rd}, k\rangle$*, then we have two cases. If* $k = \mathtt{3}$*, then the situation is analogous to previous case. On the contrary, if* $k \neq \mathtt{3}$*, we first note that* $\mathcal{S}_{GC}(\mathtt{G}^{\langle \mathtt{rd}, k\rangle}) = \emptyset$ *because* $\mathcal{S}_{GC}$ *imposes the return value of every read operation to coincide with the number of observed reads. Hence, no transition from* $\langle \mathtt{G}, \mathtt{P}\rangle$ *can be labelled by* $\langle \mathtt{rd}, k\rangle$ *if* $k \neq \mathtt{3}$ *(i.e., no read operation can return a value different from* $\mathtt{3}$ *when the replica knows three increments).*

A *multi-replica* LTS models multiple replica evolving concurrently. Suppose we have two replica, and the current state for each is $\langle \mathtt{G}_i, \mathtt{P}_i\rangle$. We assume that $\mathtt{G}_1$ and $\mathtt{G}_2$ are *compatible*, i.e., shared events have the same labels and predecessors in the two graphs (that is, $\lambda_{\mathtt{G}_1}(\mathsf{e}) = \lambda_{\mathtt{G}_2}(\mathsf{e})$ and $\{\mathsf{e}' \mid \mathsf{e}' \prec^*_{\mathtt{G}_1} \mathsf{e}\} = \{\mathsf{e}'' \mid \mathsf{e}'' \prec^*_{\mathtt{G}_2} \mathsf{e}\}$ for all $\mathsf{e} \in \mathcal{E}_{\mathtt{G}_1} \cap \mathcal{E}_{\mathtt{G}_2}$). We denote by $\mathtt{G}_1 \sqcup \mathtt{G}_2$ the union of compatible graphs. A *replicated state* is of the form $\langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P}\rangle$, where $\mathtt{P} \in \mathtt{P}_1 \otimes \mathtt{P}_2$, i.e., $\mathtt{P}$ is obtained by "synchronising" the individual arbitrations.

Definition 2.12 (Multi-replica LTS)**.** *Given a specification $\mathcal{S}$, its* multi-replica LTS *is derived from the one-replica LTS by adding the following inference rule*

$$(\textsc{Comp})\ \frac{\langle \mathtt{G}_1, \mathtt{P}|_{\mathcal{E}_{\mathtt{G}_1}}\rangle \xrightarrow{\ell} \langle \mathtt{G}'_1, \mathtt{P}'_1\rangle \qquad \mathtt{P}' \in \mathtt{P} \otimes \mathtt{P}'_1 \qquad (\mathcal{E}_{\mathtt{G}'_1} \setminus \mathcal{E}_{\mathtt{G}_1}) \cap \mathcal{E}_{\mathtt{G}_2} = \emptyset}{\langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P}\rangle \xrightarrow{\ell} \langle \mathtt{G}'_1 \sqcup \mathtt{G}_2, \mathtt{P}'\rangle}$$

The rule says that transitions from a replicated state $\langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P}\rangle$ are obtained from those of a single component $\langle \mathtt{G}_1, \mathtt{P}|_{\mathcal{E}_{\mathtt{G}_1}}\rangle$ (note that the projection recovers the local arbitration). After the transition, the replicated state evolves to the composition of the part that has not changed (namely $\mathtt{G}_2$) and the new local state $\langle \mathtt{G}'_1, \mathtt{P}'_1\rangle$. In short, global computations arise from computations of single replicas. The last condition in the premise of (Comp) ensures that events generated during computation are globally unique, i.e., new events in $\mathtt{G}'_1$ should not be already present in $\mathtt{G}_2$.

Example 2.13. *Consider again the* RDT GCounter *and a scenario similar to the one in Example 2.11, in which a replica $\mathtt{r}_1$ has performed two sequential increments and $\mathtt{r}_2$ one, but none of them has propagated its local information yet. Then, the state of the system can be represented as the composition of the states of the two replicas. Since* $\mathtt{P}$ *is uninteresting for* GCounter*, we will just focus on the visibility graph* $\mathtt{G}$*, which is depicted as follows*

$$\mathtt{G} = \mathtt{G}_1 \sqcup \mathtt{G}_2$$

$$\begin{tikzcd}
\mathtt{G}_1 & \mathtt{G}_2 \\
\langle \mathtt{inc}, ok\rangle \arrow[d] & \langle \mathtt{inc}, ok\rangle \\
\langle \mathtt{inc}, ok\rangle &
\end{tikzcd}$$

Note that $\mathtt{G}$ *combines the visibility relations corresponding to each of the replicas (which are depicted by dotted boxes).*

As discussed in Example 2.11, the one-replica LTS describes the behaviour of each operation by considering all events in the state, i.e., each fresh event dominates all other events in the visibility relation. In our scenario, this means that a read operation over $\mathtt{G}$ *can only return* $\mathtt{3}$ *(according to the one-replica LTS). However, a read operation executed over* $\mathtt{r}_1$*, which is aware of two increments, should return* $\mathtt{2}$*; and analogously, a read over* $\mathtt{r}_2$ *should return* $\mathtt{1}$.

*The multi-replica LTS accounts for the execution of operations that have a partial view of the state of the system. This is achieved via the rule (*Comp*), which accounts for decompositions of the global state. For instance, (*Comp*) allows us to derive a transition describing the evolution of the system when a read operation is performed over the replica* $\mathtt{r}_1$*. In fact, by the one-replica LTS,* *we can derive a transition from* $\langle \mathtt{G}_1, \mathtt{P}_1 \rangle$ *to* $\langle \mathtt{G}'_1, \mathtt{P}'_1 \rangle$*, which extends* $\mathtt{G}_1$ *with a new event labelled by* $\langle \mathtt{rd}, 2 \rangle$*. Then, (*Comp*) establishes that* $\mathtt{G} = \mathtt{G}_1 \sqcup \mathtt{G}_2$ *can evolve to* $\mathtt{G}'$ *below*

$$\mathtt{G}' = \mathtt{G}'_1 \sqcup \mathtt{G}_2$$

$$\begin{tikzcd}
\mathtt{G}'_1 & & \mathtt{G}'_2 \\
\langle \mathtt{inc}, ok \rangle \arrow[d] \arrow[ddr] & & \langle \mathtt{inc}, ok \rangle \\
\langle \mathtt{inc}, ok \rangle \arrow[dr] & & \\
& \langle \mathtt{rd}, 2 \rangle &
\end{tikzcd}$$

Similarly, a read operation over $\mathtt{r}_2$ *would extend the visibility with a new event labelled by* $\langle \mathtt{rd}, \mathtt{1} \rangle$ *that dominates the events in* $\mathtt{G}_2$ *(i.e., a read returns* $\mathtt{1}$ *if performed over* $\mathtt{r}_2$*).*

$$\mathtt{G}' = \mathtt{G}_1 \sqcup \mathtt{G}'_2$$

$$\begin{tikzcd}
\mathtt{G}_1 & \mathtt{G}'_2 & \\
\langle \mathtt{inc}, ok \rangle \arrow[d] & \langle \mathtt{inc}, ok \rangle \arrow[r] & \langle \mathtt{rd}, \mathtt{1} \rangle \\
\langle \mathtt{inc}, ok \rangle & &
\end{tikzcd}$$

*As highlighted in Example 2.11, we do not explicitly represent replicas. Consequently, rule (*Comp*) accounts for the behaviour of operations performed over* any *partial view of* $\mathtt{G}$*.*

For instance, consider the decomposition $\mathtt{G} = \emptyset \sqcup \mathtt{G}$*. The rule (*Comp*) can be applied to such decomposition to account for the behaviour of an operation performed over the empty state* $\emptyset$*. Intuitively, this is the state of a replica that has not performed any operation and has not received any information from the other replicas, so a read operation will return* $\mathtt{0}$*. Analogously, if we consider* $\mathtt{G} = \mathtt{G} \sqcup \mathtt{G}$*, (*Comp*) will account for operations executed over a replica that has complete view of the replicated state. Considering all possible decompositions will yield transitions in the multi-replica LTS accounting for the fact that a read operation over* $\mathtt{G}$ *may return any of the values in* $\{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}\}$*.*

3. Categorical model of specifications

In this section we first introduce the key ingredients of our categorical approach, and then we present our categorical model of coherent specifications. We assume familiarity with basic concepts of category theory, such as functors, natural transformations, limits and colimits.

3.1. Categories of relations

We first introduce a category of binary relations that will serve as the basic category for modelling both visibilities and arbitrations.

Definition 3.1 (Binary Relations)**.** *Given a finite set* $\mathtt{E}$*, a (binary)* relation ρ *over* $\mathtt{E}$*, written* $\langle \mathtt{E}, \rho\rangle$*, is a subset* $\rho \subseteq \mathtt{E} \times \mathtt{E}$*. A subset* $\mathtt{E}' \subseteq \mathtt{E}$ *is* downward closed *with respect to* ρ *if* $(\mathtt{e}, \mathtt{e}') \in \rho$ *implies* $\mathtt{e} \in \mathtt{E}'$*, for all* $\mathtt{e}' \in \mathtt{E}'$*.*

We write $\emptyset$ for the empty relation and $\mathtt{e}\ \rho\ \mathtt{e}'$ to mean $(\mathtt{e}, \mathtt{e}') \in \rho$. And we write $[\mathtt{e}]_\rho$ for the smallest downward closed set with respect to ρ including $e \in \mathtt{E}$, omitting the subscript ρ if clear from the context: It coincides with $\{\mathtt{e}' \mid \mathtt{e}'\ \rho^*\ \mathtt{e}\}$.

We now look at morphisms between relations. Besides ordinary relation-preserving morphisms, we need a special type of morphism, called *past-reflecting (pr-)morphism*, which intuitively adds no dependencies in the past of an event. As we shall see, pr-morphisms play a key role in defining the category of visibilities.

Definition 3.2 ((Binary Relation) Morphisms)**.** *A (binary relation) morphism* $\mathtt{f} : \langle \mathtt{E}, \rho\rangle \to \langle \mathtt{T}, \gamma\rangle$ *is a function* $\mathtt{f} : \mathtt{E} \to \mathtt{T}$ *such that* $\mathtt{e}\ \rho\ \mathtt{e}'$ *implies* $\mathtt{f}(\mathtt{e})\ \gamma\ \mathtt{f}(\mathtt{e}')$ *for all* $\mathtt{e}, \mathtt{e}' \in \mathtt{E}$*. A morphism* $\mathtt{f} : \langle \mathtt{E}, \rho\rangle \to \langle \mathtt{T}, \gamma\rangle$ *is* past-reflecting *(shortly, pr-morphism) if* $\mathtt{t}\ \gamma\ \mathtt{f}(\mathtt{e})$ *implies that there is* $\mathtt{e}' \in \mathtt{E}$ *such that* $\mathtt{e}'\ \rho\ \mathtt{e}$ *and* $\mathtt{t} = \mathtt{f}(\mathtt{e}')$ *for all* $\mathtt{e} \in \mathtt{E}$ *and* $\mathtt{t} \in \mathtt{T}$*.*

Past-reflecting morphisms are known under various names in different contexts, e.g. as bounded morphisms in modal logic (see, e.g., [3, Pag. 60]). We now give a useful characterisation result.

Lemma 3.3 (Characterising pr-morphisms)**.** *Let* $\mathtt{f} : \langle \mathtt{E}, \rho\rangle \to \langle \mathtt{T}, \gamma\rangle$ *be a morphism. If*

1. $\mathtt{f}(\mathtt{e})\ \gamma\ \mathtt{f}(\mathtt{e}')$ *implies* $\mathtt{e}\ \rho\ \mathtt{e}'$*, and*
2. $\bigcup_{\mathtt{e}\in\mathtt{E}} \mathtt{f}(\mathtt{e})$ *is downward closed,*

then it is a pr-morphism. If $\mathtt{f}$ *is injective, then the converse holds.*

Proof. For $\Rightarrow$), let us take $\mathtt{e} \in \mathtt{E}$ and $\mathtt{t} \in \mathtt{T}$. If $\mathtt{t}\ \gamma\ \mathtt{f}(\mathtt{e})$, then there exists $\mathtt{e}' \in \mathtt{E}$ such that $\mathtt{t} = \mathtt{f}(\mathtt{e}')$ because of (2). By (1), $\mathtt{f}(\mathtt{e}')\ \gamma\ \mathtt{f}(\mathtt{e})$ implies $\mathtt{e}'\ \rho\ \mathtt{e}$.

For $\Leftarrow$), by the definition of pr-morphism $\mathtt{f}(\mathtt{e})\ \gamma\ \mathtt{f}(\mathtt{e}')$ implies $\exists \bar{\mathtt{e}} \in \mathtt{E}.\ \bar{\mathtt{e}}\ \rho\ \mathtt{e}' \wedge \mathtt{f}(\mathtt{e}) = \mathtt{f}(\bar{\mathtt{e}})$. Since $\mathtt{f}$ is injective, $\bar{\mathtt{e}} = \mathtt{e}$ and hence $\mathtt{e}\ \rho\ \mathtt{e}'$. So, let $\mathcal{T} = \bigcup_{\mathtt{e}\in\mathtt{E}} \mathtt{f}(\mathtt{e})$. We want to show that

$$\forall \mathtt{t} \in \mathtt{T}, \mathtt{t}' \in \mathcal{T}.\ \mathtt{t}\gamma\ \mathtt{t}' \text{ implies } \mathtt{t} \in \mathcal{T}$$

The proof follows by contradiction. Assume that $\exists \mathtt{t} \in \mathtt{T}, \mathtt{t}' \in \mathcal{T}.\ \mathtt{t}\ \gamma\ \mathtt{t}' \wedge \mathtt{t} \notin \mathcal{T}$. By definition of $\mathcal{T}$, $\exists \mathtt{e} \in \mathtt{E}$ such that $\mathtt{f}(\mathtt{e}) = \mathtt{t}'$. Since $\mathtt{f}$ is a pr-morphism, then

$$\mathtt{t}\ \gamma\ \mathtt{f}(\mathtt{e}) \text{ implies } \exists \mathtt{e}' \in \mathtt{E}.\ \mathtt{e}'\ \rho\ \mathtt{e}\ \wedge\ \mathtt{t} = \mathtt{f}(\mathtt{e}')$$

Therefore $\mathtt{t} = \mathtt{f}(\mathtt{e}') \in \mathcal{T}$, which contradicts the assumption $\mathtt{t} \notin \mathcal{T}$. □

Both classes of morphisms are closed under composition: **Bin** denotes the category of relations and their morphisms and **PBin** the sub-category of pr-morphisms. The category **Bin** has finite limits and finite colimits, which are computed point-wise as in **Set**. The structure is lifted by and large to **PBin**: Finite colimits and binary pullbacks in **PBin** are computed as in **Bin**. Yet there is no terminal object, as clearly not all relations admit a past-reflecting morphism into the singleton.

Monos in **Bin** are just morphisms whose underlying function is injective, and similarly in **PBin**, so that the inclusion functor preserves (and reflects) them. Thus, we can lift the following lemma from the category of sets and functions.

Lemma 3.4 (Monos under pushouts)**.** *Pushouts in* **Bin** *(and thus in* **PBin***) preserve monos.*

Given a set of labels $\mathcal{L}$, the category of labelled relations is $\mathbf{Bin}(\mathcal{L})$.

Definition 3.5 (Category of labelled relations)**.** *The category* $\mathbf{Bin}(\mathcal{L})$ *is defined as the comma category* $\mathtt{U_r} \downarrow \mathcal{L}$*, where* $\mathtt{U_r} : \mathbf{Bin} \to \mathbf{Set}$ *is the forgetful functor, mapping a binary relation to its underlying set of elements. Explicitly, an object in* $\mathbf{Bin}(\mathcal{L})$ *is a triple* $(\mathtt{E}, \rho, \lambda)$ *for a labeling function* $\lambda : \mathtt{E} \to \mathcal{L}$*. A label-preserving morphism* $(\mathtt{E}, \rho, \lambda) \to (\mathtt{E}', \rho', \lambda')$ *is a morphism* $\mathtt{f} : (\mathtt{E}, \rho) \to (\mathtt{E}', \rho')$ *such that* $\lambda(\mathtt{s}) = \lambda'(\mathtt{f}(\mathtt{s}))$ *for all* $\mathtt{s} \in \mathtt{E}$.

The category $\mathbf{PBin}(\mathcal{L})$ is defined analogously, with the requirement that the morphisms are past-reflecting. In both categories, finite colimits and binary pullbacks always exist and are essentially computed as in **Bin**.

3.2. Categories of Graphs and Paths

We now move to introduce specific sub-categories that are going to be used for both the syntax and the semantics of specifications.

Definition 3.6 (**PDag**)**.** **PDag** *is the full sub-category of* **PBin** *whose objects are acyclic graphs (relations whose transitive closures is a strict partial order).*

The full sub-category of **Bin** whose objects are acyclic graphs is not suited for our purposes, since it does not admit pushouts, not even along monos. Pushouts are important for our work, as they will allow us to generalise composition of visibilities. The sub-category of **Bin** with pr-morphisms is much more well-behaved, still remaining computationally simple. In fact, it enjoys the following property. Recall that a functor *reflects* (co)limits if every diagram mapped to a (co)limit by the functor is a (co)limit itself.

Proposition 3.7 (Properties of **PDag**)**.** *The inclusion functor* $\mathbf{PDag} \to \mathbf{PBin}$ *reflects finite colimits and binary pullbacks.*

In other terms, finite colimits and binary pullbacks in **PDag** are computed as in **PBin**, hence as in **Bin**.

We now move to consider the category of paths.

$$\begin{tikzcd}
\langle \mathtt{wr}(1), ok\rangle \arrow[d] \\
\langle \mathtt{wr}(2), ok\rangle \arrow[d] \\
\langle \mathtt{rd}, \{2\}\rangle
\end{tikzcd}$$

(a) $\mathtt{G}_1$

$$\begin{tikzcd}
\langle \mathtt{wr}(1), ok\rangle \arrow[d] \arrow[dd, bend right] \\
\langle \mathtt{wr}(2), ok\rangle \arrow[d] \\
\langle \mathtt{rd}, \{2\}\rangle
\end{tikzcd}$$

(b) $\mathtt{P}_1$

$$\begin{tikzcd}
\langle \mathtt{wr}(1), ok\rangle \arrow[d] \arrow[dd, bend right] & \langle \mathtt{wr}(3), ok\rangle \\
\langle \mathtt{wr}(2), ok\rangle \arrow[d] & \\
\langle \mathtt{rd}, \{2\}\rangle &
\end{tikzcd}$$

(c) $\mathtt{G}_2$

$$\begin{tikzcd}
\langle \mathtt{wr}(1), ok\rangle \arrow[d] \arrow[dd, bend right] \arrow[ddr] & \langle \mathtt{wr}(3), ok\rangle \arrow[dd] \\
\langle \mathtt{wr}(2), ok\rangle \arrow[d] \arrow[dr] & \\
\langle \mathtt{rd}, \{2\}\rangle \arrow[r, dashed] & \langle \mathtt{rd}, \{2,3\}\rangle
\end{tikzcd}$$

(d) $\mathtt{G}_3$

Figure 4: Some labelled graphs

Definition 3.8 (**Path**). ***Path** is the full sub-category of* **Bin** *whose objects are paths (i.e., total orders).*

Note that defining **Path** as only containing pr-morphisms would be too restrictive, since there exists a pr-morphism between two paths if and only if one path is a prefix of the other. Nevertheless, as for **PDag**, finite colimits in **Path** are computed as in **Bin**.

Proposition 3.9 (Properties of **Path**). *The inclusion functor* **Path** $\to$ **Bin** *reflects finite colimits.*

As for relations, we consider suitable comma categories in order to capture labelled paths and graphs. In particular, we use the forgetful functors $\mathtt{U}_{\mathtt{rp}}$: **Path** $\to$ **Set** and $\mathtt{U}_{\mathtt{pd}}$: **PDag** $\to$ **Set**: For a set of labels $\mathcal{L}$ we denote $\mathbf{PDag}(\mathcal{L}) = \mathtt{U}_{\mathtt{rp}} \downarrow \mathcal{L}$ and $\mathbf{Path}(\mathcal{L}) = \mathtt{U}_{\mathtt{pd}} \downarrow \mathcal{L}$. Once more, finite colimits and binary pullbacks always exist and are essentially computed as in **Bin**.

Example 3.10. *We illustrate some labelled graphs in Fig. 4 and remark that* $\mathtt{P}_1$ *is the only path in the figure. Note that* $\mathtt{G}_1$ *is not a path because the relation is not transitive. There is an obvious label-preserving morphism* $\mathtt{f}_1 : \mathtt{G}_1 \to \mathtt{P}_1$, *but this is not a pr-morphism because the edge from* $\langle \mathtt{wr}(1), ok\rangle$ *to* $\langle \mathtt{rd}, \{2\}\rangle$ *in* $\mathtt{P}_1$ *is not matched in* $\mathtt{G}_1$. *On the contrary, there is no morphism from* $\mathtt{P}_1$ *to* $\mathtt{G}_1$. *Note that the label-preserving morphisms* $\mathtt{f}_2 : \mathtt{P}_1 \to \mathtt{G}_2$ *and* $\mathtt{f}_3 : \mathtt{G}_2 \to \mathtt{G}_3$ *are pr-morphisms (and consequently,* $\mathtt{f}_2;\mathtt{f}_3 : \mathtt{P}_1 \to \mathtt{G}_3$ *is so).*

3.3. Category of visibilities

We now study the category of visibility relations. We first introduce an operation that will be handy for our categorical characterisation. We say that a graph $\mathtt{G}$ is *rooted* if there exists a (necessarily unique) event $\mathtt{e} \in \mathcal{E}_{\mathtt{G}}$ such that $\mathtt{G} = \mathtt{G}|_{[\mathtt{e}]}$.

Definition 3.11 (Extension). *Let* $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$ *be a mono pr-morphism. It is an* extension *along* ℓ *(shortly,* ℓ-extension*) if, for a fresh event* $\top$, $\mathcal{E}_{\mathtt{G}_2} = \mathcal{E}_{\mathtt{G}_1} \cup \{\top\}$, $\mathtt{f} : \mathcal{E}_{\mathtt{G}_1} \to \mathcal{E}_{\mathtt{G}_1} \cup \{\top\}$ *is the associated injection, and* $\lambda_{\mathtt{G}_2}(\top) = \ell$.

Intuitively, $\mathtt{G}_2$ is obtained by adding to the visibility relation $\mathtt{G}_1$ a new event "seeing" some events in $\mathtt{G}_1$. We say that an extension $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$ is a *root* extension if $\mathtt{G}_2$ is rooted.

Example 3.12. *Noting that* $\mathtt{f}_2$ *and* $\mathtt{f}_3$ *in Ex. 3.10 are mono, the former is an extension along* $\langle \mathtt{wr}(3), ok\rangle$ *and the latter is an extension along* $\langle \mathtt{rd}, \{2,3\}\rangle$*. Since* $\mathtt{G}_3$ *is rooted,* $\mathtt{f}_3$ *is a rooted extension; however,* $\mathtt{f}_2$ *is not rooted because* $\mathtt{G}_2$ *is not. Note that* $\mathtt{f}_2;\mathtt{f}_3$ *is a pr-morphism but not an extension because the target* $\mathtt{G}_3$ *adds two new events to the source* $\mathtt{P}_1$*.*

Proposition 3.13. *Rooted graphs form a family of separators of* $\mathbf{PDag}(\mathcal{L})$*.*

Proof. We need to show that for any pair of pr-morphisms $\mathtt{f}_1, \mathtt{f}_2 : \mathtt{G}_1 \to \mathtt{G}_2$ such that $\mathtt{f}_1 \neq \mathtt{f}_2$ there is a rooted graph $\mathtt{G}$ and a morphism $\mathtt{f} : \mathtt{G} \to \mathtt{G}_1$ such that $\mathtt{f};\mathtt{f}_1 \neq \mathtt{f};\mathtt{f}_2$. Given $\mathtt{e} \in \mathcal{E}_{\mathtt{G}_1}$ such that $\mathtt{f}_1(\mathtt{e}) \neq \mathtt{f}_2(\mathtt{e})$, it suffices to consider the pr-morphism $\mathtt{f} : \mathtt{G}_1|_{[\mathtt{e}]} \to \mathtt{G}_1$. □

We now further curb the arrows in $\mathbf{PDag}(\mathcal{L})$ to *monic* ones. Intuitively, we are only interested in what happens if we add further events to visibility relations. We thus consider the sub-category $\mathbf{PIDag}(\mathcal{L})$ of acyclic graphs and monic pr-morphisms. Note that the chosen morphism $\mathtt{f}$ in the proof of Proposition 3.13 is monic, since a morphism in $\mathbf{PDag}(\mathcal{L})$ is monic if and only if the underlying function is injective.

We can then show that rooted graphs are also a family of generators for the sub-category $\mathbf{PIDag}(\mathcal{L})$. Before giving the result, we observe that, as is the case for the category of sets and injective functions, $\mathbf{PIDag}(\mathcal{L})$ lacks pushouts. However, the following simple technical result, analogous to Lemma 3.4, allows us to compute those pushouts in the larger category $\mathbf{PDag}(\mathcal{L})$

Lemma 3.14 (Monos under pushouts, 2)**.** *Pushouts in* $\mathbf{PDag}(\mathcal{L})$ *preserve monos.*

We can now state the announced important characterisation of $\mathbf{PIDag}(\mathcal{L})$.

Proposition 3.15. $\mathbf{PIDag}(\mathcal{L})$ *is the smallest sub-category of* $\mathbf{PDag}(\mathcal{L})$ *containing all root extension morphisms and closed under finite colimits.*

Proof. First, note that, since pushouts in $\mathbf{PDag}(\mathcal{L})$ preserve monos, the smallest sub-category of $\mathbf{PDag}(\mathcal{L})$ containing all root extensions and closed under finite colimits is surely a sub-category of $\mathbf{PIDag}(\mathcal{L})$. So, given a monic pr-morphism $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$, we need to prove that it can be generated from root extension morphisms via colimits. We proceed by induction on the cardinality of $\mathcal{E}_{\mathtt{G}_2}$.

If the cardinality is 0, then $\mathtt{f}$ must be the identity of the empty graph. Otherwise, consider $\mathtt{G}_2$ and assume that it is rooted with root $\mathtt{e}$. Now, if $\mathtt{e} \in \mathsf{img}(\mathtt{f})$, since the image of a pr-morphism is downward closed, it turns out that $\mathtt{f}$ is the identity of $\mathtt{G}_2$. If it is not in the image, then $\mathtt{f}$ can be decomposed as $\mathtt{G}_1 \to (\mathtt{G}_2 \setminus \mathtt{e}) \to \mathtt{G}_2$: the left-most is given by induction, while the right-most is a root extension morphism.

Without loss of generality, let us assume now that $\mathtt{G}_2$ has two distinct roots, namely $\mathtt{e}_1$ and $\mathtt{e}_2$, and that the image of $\mathtt{f}$ is contained in $\mathtt{G}_2|_{[\mathtt{e}_1]}$. Now, $\mathtt{f}$ can be decomposed as $\mathtt{G}_1 \to \mathtt{G}_2|_{[\mathtt{e}_1]} \to \mathtt{G}_2$: the left-most is given by induction, while the right-most is obtained via the pushout of the span $\mathtt{G}_2|_{[\mathtt{e}_1]} \cap \mathtt{G}_2|_{[\mathtt{e}_2]} \to \mathtt{G}_2|_{[\mathtt{e}_1]}$. □

3.4. Category of arbitrations

We know from Section 2.4 that the objects of our category of arbitrations must be compatible sets of paths. We now turn our attention to morphisms. We first look at two constructions, called *saturation* and *retraction*, which act on arbitrations by extending or restricting them, respectively, in a maximal way, according to a function between sets of events. These constructions will be used to define a notion of morphisms of arbitrations, and to describe limits and colimits in the ensuing category.

Definition 3.16 (Path saturation)**.** *Let* $\mathtt{P}$ *be a path and* $\mathtt{f} : (\mathcal{E}_\mathtt{P}, \lambda_\mathtt{P}) \to (\mathcal{E}, \lambda)$ *a label-preserving function. The saturation of* $\mathtt{P}$ *along* $\mathtt{f}$ *is defined as*

$$\mathtt{sat}(\mathtt{P}, \mathtt{f}) = \{\mathtt{Q} \mid \mathtt{Q} \in \mathbb{P}(\mathcal{E}, \lambda) \textit{ and } \mathtt{f} \textit{ induces a morphism } \mathtt{f} : \mathtt{P} \to \mathtt{Q} \textit{ in } \mathbf{Path}(\mathcal{L})\}$$

Saturation is generalised to sets of paths $\mathcal{X} \subseteq \mathbb{P}(\mathcal{E}, \lambda)$ *as* $\bigcup_{\mathtt{P} \in \mathcal{X}} \mathtt{sat}(\mathtt{P}, \mathtt{f})$*.*

Note that, should $\mathtt{f}$ not be injective, it could be that $\mathtt{sat}(\mathtt{P}, \mathtt{f}) = \emptyset$.

Example 3.17. *Consider the injective, label-preserving function* $\mathtt{f}$ *from set* $\{\langle \mathtt{wr}(1), ok\rangle, \langle \mathtt{wr}(2), ok\rangle\}$ *to set* $\{\langle \mathtt{wr}(1), ok\rangle, \langle \mathtt{wr}(2), ok\rangle, \langle \mathtt{rd}, 2\rangle\}$*. Then, we have*

$$\mathtt{sat}\left(\begin{array}{c}\langle \mathtt{wr}(1), ok\rangle\\ | \\ \langle \mathtt{wr}(2), ok\rangle\end{array}, \mathtt{f}\right) = \left\{\begin{array}{ccc}\langle \mathtt{wr}(1), ok\rangle & \langle \mathtt{wr}(1), ok\rangle & \langle \mathtt{rd}, 2\rangle\}\\ | & | & | \\ \langle \mathtt{wr}(2), ok\rangle\ , & \langle \mathtt{rd}, 2\rangle\} & ,\ \langle \mathtt{wr}(1), ok\rangle \\ | & | & | \\ \langle \mathtt{rd}, 2\rangle\} & \langle \mathtt{wr}(2), ok\rangle & \langle \mathtt{wr}(2), ok\rangle\end{array}\right\}$$

Intuitively, saturation adds $\langle \mathtt{rd}, 2\rangle$ *– and in general those events that are not in the image of* $\mathtt{f}$ *– to the original path in all possible ways, preserving the order of the original events.*

Definition 3.18 (Path retraction)**.** *Let* $\mathtt{Q}$ *be a path and* $\mathtt{f} : \mathcal{E} \to \mathcal{E}_\mathtt{Q}$ *a function. The retraction of* $\mathtt{Q}$ *along* $\mathtt{f}$ *is defined as*

$$\mathtt{ret}(\mathtt{Q}, \mathtt{f}) = \{\mathtt{P} \mid \mathtt{P} \in \mathbb{P}(\mathcal{E}, \lambda) \textit{ and } \mathtt{f} \textit{ induces a morphism } \mathtt{f} : \mathtt{P} \to \mathtt{Q} \textit{ in } \mathbf{Path}(\mathcal{L})\}$$

The notion of retraction is extended to sets of paths $\mathcal{X} \subseteq \mathbb{P}(\mathcal{E}, \lambda)$ *as* $\bigcup_{\mathtt{Q} \in \mathcal{X}} \mathtt{ret}(\mathtt{Q}, \mathtt{f})$*.*

Note that λ is fully characterised as the restriction of $\lambda_\mathtt{Q}$ along the mapping. Should $\mathtt{f}$ be injective, $\mathtt{ret}(\mathtt{Q}, \mathtt{f})$ would be a singleton, and if $\mathtt{f}$ is an inclusion, then $\mathtt{ret}(\mathtt{Q}, \mathtt{f}) = \mathtt{Q}|_\mathcal{E}$.

We may now start considering the relationship between the two notions.

Lemma 3.19. *Let* $\mathcal{X}_1 \subseteq \mathbb{P}(\mathcal{E}_1, \lambda_1)$ *be a set of paths and* $\mathtt{f} : (\mathcal{E}_1, \lambda_1) \to (\mathcal{E}_2, \lambda_2)$ *a label-preserving function. Then* $\mathcal{X}_1 \subseteq \mathtt{ret}(\mathtt{sat}(\mathcal{X}_1, \mathtt{f}), \mathtt{f})$*. If* $\mathtt{f}$ *is injective, then the equality holds.*

Lemma 3.20. *Let $\mathcal{X}_2 \subseteq \mathbb{P}(\mathcal{E}_2, \lambda_2)$ be a set of paths and $\mathtt{f} : \mathcal{E}_1 \to \mathcal{E}_2$ a function. Then $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathtt{ret}(\mathcal{X}_2, \mathtt{f}), \mathtt{f})$.*

Here, injectiveness does not ensure that the equality holds. We say that an injective function $\mathtt{f}$ is *saturated* with respect to $\mathcal{X}_2$ if the equality does hold.

Example 3.21. *Consider the set of paths $\mathcal{X}_1$ and $\mathcal{X}_2$ and the pr-morphism $\mathtt{f}$ below*

$$\mathcal{X}_1 = \left\{ \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \end{array} \right\} \qquad \mathcal{X}_2 = \left\{ \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array} \right\} \qquad \mathtt{f}: \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \end{array} \to \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array}$$

the underlying function $\mathtt{f}$ (defined in Example 3.17) is not *saturated with respect to $\mathcal{X}_2$ because*

$$\left\{ \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array} \right\} \neq \mathtt{sat}(\mathtt{ret}(\left\{ \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array} \right\}, \mathtt{f}), \mathtt{f}) = \mathtt{sat}(\left\{ \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \end{array} \right\}, \mathtt{f})$$

We can now exploit saturation to get a simple definition of our category of arbitrations.

Definition 3.22 (ps-morphism). *Let $\mathcal{X}_1 \subseteq \mathbb{P}(\mathcal{E}_1, \lambda_1)$ and $\mathcal{X}_2 \subseteq \mathbb{P}(\mathcal{E}_2, \lambda_2)$ be sets of paths. A path-set morphism (shortly, ps-morphism) $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ is a label-preserving function $\mathtt{f} : (\mathcal{E}_1, \lambda_1) \to (\mathcal{E}_2, \lambda_2)$ such that $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathcal{X}_1, \mathtt{f})$.*

Intuitively, there is a ps-morphism from the set of paths $\mathcal{X}_1$ to the set of paths $\mathcal{X}_2$ if any path in $\mathcal{X}_2$ can be obtained by adding events to some path in $\mathcal{X}_1$. This notion captures the idea that arbitrations of larger visibilities are obtained as extensions of smaller visibilities. We say that mono ps-morphism $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ is a *ps-extension* (along ℓ) if its underlying function between events is of the form $\mathtt{f} : \mathcal{E}_{\mathcal{X}_1} \to \mathcal{E}_{\mathcal{X}_1} \cup \{\top\}$, with ℓ labelling the fresh event $\top$.

Example 3.23. *Consider the following three sets and the function $\mathtt{f}$ from Example 3.17*

$$\mathcal{X}_1 = \left\{ \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \end{array} \right\} \quad \mathcal{X}_2 = \left\{ \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array}, \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \end{array} \right\} \quad \mathcal{X}_3 = \left\{ \begin{array}{c} \langle \mathtt{wr(1)}, ok\rangle \\ | \\ \langle \mathtt{wr(2)}, ok\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array}, \begin{array}{c} \langle \mathtt{wr(2)}, ok\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \\ | \\ \langle \mathtt{wr(1)}, ok\rangle \end{array} \right\}$$

Now, $\mathtt{f}$ induces a ps-morphism $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ because $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathcal{X}_1, \mathtt{f})$ (the latter is shown in Example 3.17). On the contrary, there is no ps-morphism from $\mathcal{X}_1$ to $\mathcal{X}_3$: the rightmost path of $\mathcal{X}_3$ cannot be obtained by extending a path of $\mathcal{X}_1$ with an event labelled by $\langle \mathtt{rd}, 2\rangle$. Note that the ps-morphism induced by $\mathtt{f}$ is a ps-extension along $\langle \mathtt{rd}, 2\rangle$.

Definition 3.24 (Sets of Paths Category)**.** *We define* $\mathbf{SPath}(\mathcal{L})$ *as the category whose objects are sets of paths labelled over* $\mathcal{L}$ *and arrows are ps-morphisms.*

Proposition 3.25 (Properties of $\mathbf{SPath}$)**.** *The category* $\mathbf{SPath}(\mathcal{L})$ *has finite colimits along monos and binary pullbacks.*

Proof. (Strict) initial object. The (unique) initial object is $\langle\emptyset, \{\epsilon\}, \emptyset\rangle$, with $\epsilon \in \mathbb{P}(\emptyset,\emptyset)$ the empty path. Let $\mathcal{X} \subseteq \mathbb{P}(\mathcal{E},\lambda)$ and $!\colon \emptyset \to \mathcal{E}$ the unique function from the empty set to $\mathcal{E}$. We have the (unique) function $!\colon (\emptyset,\emptyset) \to (\mathcal{E},\lambda)$ such that $\mathcal{X} \subseteq \mathtt{sat}(\{\epsilon\}, !) = \mathbb{P}(\mathcal{E},\lambda)$.

Binary Pushouts. Let $\mathcal{X}, \mathcal{X}_1$, and $\mathcal{X}_2$ be sets of paths and $\mathtt{f}_i : \mathcal{X} \to \mathcal{X}_i$ ps-morphisms. Consider the underlying functions $\mathtt{f}_i : \mathcal{E} \to \mathcal{E}_i$ and their pushout $\mathtt{f}'_i : \mathcal{E}_i \to \mathcal{E}_1 +_{\mathcal{E}} \mathcal{E}_2$ in the category of sets and functions: it induces a pushout $\mathtt{f}'_i : \mathcal{X}_i \to \mathtt{sat}(\mathcal{X}_1, \mathtt{f}'_1) \cap \mathtt{sat}(\mathcal{X}_2, \mathtt{f}'_2)$ in $\mathbf{SPath}(\mathcal{L})$.

Binary Pullbacks. Let $\mathcal{X}, \mathcal{X}_1$, and $\mathcal{X}_2$ be sets of paths and $\mathtt{f}_i : \mathcal{X}_i \to \mathcal{X}$ ps-morphisms. Consider the underlying functions $\mathtt{f}_i : \mathcal{E}_i \to \mathcal{E}$ and their pullback $\mathtt{f}'_i : \mathcal{E}_1 \times_{\mathcal{E}} \mathcal{E}_2 \to \mathcal{E}$ in the category of sets: It induces a pullback $\mathtt{f}'_i : \mathtt{ret}(\mathcal{X}_1, \mathtt{f}'_1) \cup \mathtt{ret}(\mathcal{X}_2, \mathtt{f}'_2) \to \mathcal{X}_i$ in $\mathbf{SPath}(\mathcal{L})$. □

The above characterisation of pushouts is enabled by the fact that we considered injective functions. To help intuition, we now instantiate that characterisation to suitable inclusions. We obtain that pushouts of inclusions precisely amount to products of sets of paths, as defined in Definition 2.8, which is a crucial step towards characterising coherent specifications.

Lemma 3.26. *Let* $\mathtt{f}_i : \mathcal{X} \to \mathcal{X}_i$ *be ps-morphisms such that the underlying functions* $\mathtt{f}_i : \mathcal{E} \to \mathcal{E}_i$ *are inclusions and* $\mathcal{E} = \mathcal{E}_1 \cap \mathcal{E}_2$*. Then their pushout is given by* $\mathtt{f}'_i : \mathcal{X}_i \to \mathcal{X}_1 \otimes \mathcal{X}_2$*.*

Proof. By definition $\mathcal{X}_1 \otimes \mathcal{X}_2 = \{\mathtt{P} \mid \mathtt{P}$ is a path over $\bigcup_i \mathcal{E}_i$ and $\mathtt{P}|_{\mathcal{E}_i} \in \mathcal{X}_i\}$. Note also that $\mathtt{sat}(\mathcal{X}_i, \mathtt{f}'_\mathtt{i}) = \bigcup_{\mathtt{Q} \in \mathcal{X}_i} \{\mathtt{P} \mid \mathtt{P} \in \mathbb{P}(\bigcup_i \mathcal{E}_i, \bigcup_i \lambda_i)$ and $\mathtt{f}'_1$ induces a path morphism $\mathtt{f}'_\mathtt{i} : \mathtt{P} \to \mathtt{Q}\}$. Since $\mathtt{f}'_\mathtt{i}$ is an inclusion, the latter condition equals to $\mathtt{P}|_{\mathcal{E}_i} = \mathtt{Q}$, thus the property holds. □

Example 3.27. *Consider the following ps-morphisms*

$$\left\{\begin{array}{c}\langle\mathtt{wr}(1), ok\rangle\\ \mid \\ \langle\mathtt{wr}(2), ok\rangle\\ \mid \\ \langle\mathtt{rd}, 2\rangle\end{array}\right\} \leftarrow \left\{\begin{array}{c}\langle\mathtt{wr}(1), ok\rangle\\ \mid \\ \langle\mathtt{wr}(2), ok\rangle\end{array}\right\} \rightarrow \left\{\begin{array}{ccc}\langle\mathtt{wr}(1), ok\rangle & & \langle\mathtt{wr}(2), ok\rangle\\ \mid & & \mid \\ \langle\mathtt{wr}(2), ok\rangle & , & \langle\mathtt{wr}(1), ok\rangle\\ \mid & & \mid \\ \langle\mathtt{rd}, 1\rangle & & \langle\mathtt{rd}, 1\rangle\end{array}\right\}$$

then, the pushout is given by the following ps-morphisms

$$\left\{\begin{array}{c}\langle\mathtt{wr}(1), ok\rangle\\ \mid \\ \langle\mathtt{wr}(2), ok\rangle\\ \mid \\ \langle\mathtt{rd}, 2\rangle\end{array}\right\} \rightarrow \left\{\begin{array}{ccc}\langle\mathtt{wr}(1), ok\rangle & & \langle\mathtt{wr}(1), ok\rangle\\ \mid & & \mid \\ \langle\mathtt{wr}(2), ok\rangle & & \langle\mathtt{wr}(2), ok\rangle\\ \mid & , & \mid \\ \langle\mathtt{rd}, 1\rangle & & \langle\mathtt{rd}, 2\rangle\\ \mid & & \mid \\ \langle\mathtt{rd}, 2\rangle & & \langle\mathtt{rd}, 1\rangle\end{array}\right\} \leftarrow \left\{\begin{array}{ccc}\langle\mathtt{wr}(1), ok\rangle & & \langle\mathtt{wr}(2), ok\rangle\\ \mid & & \mid \\ \langle\mathtt{wr}(2), ok\rangle & , & \langle\mathtt{wr}(1), ok\rangle\\ \mid & & \mid \\ \langle\mathtt{rd}, 1\rangle & & \langle\mathtt{rd}, 1\rangle\end{array}\right\}$$

An analogous property holds for pullbacks. Let $\mathtt{f}_i : \mathcal{X}_i \to \mathcal{X}$ be ps-morphisms such that the underlying functions are inclusions: The pullback is given as $\mathtt{f}'_i : \bigcup_i \mathcal{X}_i|_{\mathcal{E}_1 \cap \mathcal{E}_2} \to \mathcal{X}_i$. In particular, the square below is both a pullback and a pushout

$$\begin{tikzcd}
\bigcup_i \mathcal{X}_i|_{\mathcal{E}_1 \cap \mathcal{E}_2} \arrow[r, hook] \arrow[d, hook] & \mathcal{X}_1 \arrow[d, hook] \\
\mathcal{X}_2 \arrow[r, hook] & \mathcal{X}_1 \otimes \mathcal{X}_2
\end{tikzcd}$$

3.5. Coherent Functors

It is now time for moving towards our categorical characterisation of coherent specifications. We first provide a simple technical result.

Lemma 3.28. *Let $\mathcal{S}$ be a coherent specification and $\mathcal{E} \subseteq \mathcal{E}_{\mathtt{G}}$. If $\mathcal{E}$ is downward closed, then $\mathcal{S}(\mathtt{G})|_{\mathcal{E}} \subseteq \mathcal{S}(\mathtt{G}|_{\mathcal{E}})$.*

Proof. Let $\mathcal{E}$ be downward closed, and note that this amounts to requiring $\mathcal{E} = \bigcup_{e \in \mathcal{E}} [\mathtt{e}]$, hence for all $\mathtt{e} \in \mathcal{E}$ we have that $(\mathtt{G}|_{\mathcal{E}})|_{[\mathtt{e}]} = \mathtt{G}|_{[\mathtt{e}]}$. By the latter and by coherence we have $\mathcal{S}(\mathtt{G})|_{\mathcal{E}} = (\bigotimes_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} \mathcal{S}(\mathtt{G}|_{[\mathtt{e}]}))\Big|_{\mathcal{E}}$ and $\mathcal{S}(\mathtt{G}|_{\mathcal{E}}) = \bigotimes_{\mathtt{e} \in \mathcal{E}} \mathcal{S}(\mathtt{G}|_{[\mathtt{e}]})$. Note that $(\bigotimes_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} \mathcal{S}(\mathtt{G}|_{[\mathtt{e}]}))\Big|_{\mathcal{E}} \subseteq \bigotimes_{\mathtt{e} \in \mathcal{E}} \mathcal{S}(\mathtt{G}|_{[\mathtt{e}]})$ because a path in the former can always be restricted to a suitable path with fewer events on the latter (the converse in general does not hold). ☐

We now introduce our functorial models of specifications, consisting of functors preserving relevant categorical structure. In the following we say that a functor $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ *weakly preserves* finite colimits if any commuting diagram in $\mathbf{PIDag}(\mathcal{L})$ that is a colimit (via the inclusion functor) in $\mathbf{PDag}(\mathcal{L})$ is mapped by $\mathbb{F}$ to a colimit in $\mathbf{SPath}(\mathcal{L})$.

Definition 3.29. *A functor $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ is* coherent *if it maps root ℓ-extensions to ℓ-extensions and weakly preserves finite colimits.*

The rest of this section is dedicated to proving that coherent functors are a sound and complete characterisation of coherent specifications. More specifically, we shall prove that coherent specifications induce coherent functors (soundness) and vice versa (completeness). To this purpose, we need the following two technical lemmas. The first one spells out two important properties of coherent functors: They preserve monos and map graphs to sets of paths of the correct type. The second one says that, for each graph $\mathtt{G}$, a coherent functor gives a set of paths which is the product of paths for sub-graphs of $\mathtt{G}$, as required by coherence (Definition 2.9).

Lemma 3.30. *Let $\mathbb{F}$ be a coherent functor. Then*

(i) $\mathbb{F}$ preserves monos;

(ii) $\mathbb{F}(\mathtt{G})$ *is a set of paths over* $(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}})$*, for all* $\mathtt{G}$ *in* $\mathbf{PIDag}(\mathcal{L})$*.*

Proof. Recall that morphisms of $\mathbf{PIDag}(\mathcal{L})$ are generated from root ℓ-extensions via finite colimits (Proposition 3.15). One can show that $\mathbb{F}$ maps $\mathtt{f}\colon \mathtt{G}_1 \hookrightarrow \mathtt{G}_2$ to a monic ps-morphism by induction on the number of vertices of $\mathtt{G}_2$, using the fact that $\mathbb{F}$ maps root ℓ-extensions to ℓ-extensions, which are monic, and weakly preserves finite colimits, hence it maps pushouts of monos to pushouts of monos. Point *(ii)* can be shown via an analogous argument, noting that $\mathbb{F}$ preserves the label of the additional event when mapping root ℓ-extensions to ℓ-extensions. □

Lemma 3.31. *Let* $\mathbb{F}$ *be a coherent functor and* $\mathtt{f}_i\colon \mathtt{G} \hookrightarrow \mathtt{G}_i$ $(i = 1, 2)$ *mono pr-morphisms such that* $\mathcal{E}_{\mathtt{G}} = \mathcal{E}_{\mathtt{G}_1} \cap \mathcal{E}_{\mathtt{G}_2}$*. Then there exists a pushout in* $\mathbf{SPath}(\mathcal{L})$

$$\begin{tikzcd}
\mathbb{F}(\mathtt{G}) \arrow[r, hook, "\mathbb{F}(\mathtt{f}_1)"] \arrow[d, hook, "\mathbb{F}(\mathtt{f}_2)"'] & \mathbb{F}(\mathtt{G}_1) \arrow[d, hook] \\
\mathbb{F}(\mathtt{G}_2) \arrow[r, hook] & \mathbb{F}(\mathtt{G}_1) \otimes \mathbb{F}(\mathtt{G}_2)
\end{tikzcd}$$

Proof. $\mathbb{F}$ preserves monos, by Lemma 3.30, so $\mathbb{F}(\mathtt{f}_i)\colon \mathbb{F}(\mathtt{G}) \hookrightarrow \mathbb{F}(\mathtt{G}_i)$ are monos and by Lemma 3.26 their pushout is the diagram above. □

We now state our soundness result.

Theorem 3.32 (Soundness)**.** *A coherent specification* $\mathcal{S}$ *induces a coherent functor* $\mathbb{M}(\mathcal{S}) : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$*.*

Proof. For $\mathtt{G}$ we define $\mathbb{M}(\mathcal{S})(\mathtt{G})$ as $\mathcal{S}(\mathtt{G})$ and for $\mathtt{f} : \mathtt{G} \to \mathtt{G}'$ we define $\mathbb{M}(\mathcal{S})(\mathtt{f})$ as the ps-morphism with underlying injective function $\mathtt{f}\colon (\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}}) \hookrightarrow (\mathcal{E}_{\mathtt{G}'}, \lambda_{\mathtt{G}'})$.

We first show that $\mathtt{f}$ is a valid ps-morphism from $\mathcal{S}(\mathtt{G})$ into $\mathcal{S}(\mathtt{G}')$, i.e., $\mathcal{S}(\mathtt{G}') \subseteq \mathtt{sat}(\mathcal{S}(\mathtt{G}), \mathtt{f})$ and, since we are considering specifications preserving isomorphisms, we can restrict our attention to the case where $\mathtt{f}$ is an inclusion. Since $\mathtt{f}$ is a pr-morphism, $\bigcup_{\mathtt{e}\in\mathcal{E}_{\mathtt{G}}} \mathtt{f}(\mathtt{e})$ is downward-closed in $\mathtt{G}'$ and thus by Lemma 3.28 we have $\mathcal{S}(\mathtt{G}')|_{\mathcal{E}_{\mathtt{G}}} \subseteq \mathcal{S}(\mathtt{G}'|_{\mathcal{E}_{\mathtt{G}}}) = \mathcal{S}(\mathtt{G})$, the latter equality given by coherence. Now, consider a path $\mathtt{P} \in \mathcal{S}(\mathtt{G}')$. Since $\mathtt{P}|_{\mathcal{E}_{\mathtt{G}}} \in \mathcal{S}(\mathtt{G})$, we have $\mathtt{P} \in \mathtt{sat}(\mathcal{S}(\mathtt{G}), \mathtt{f})$, because saturation adds missing events – namely those in $\mathcal{E}_{\mathtt{G}'} \setminus \mathcal{E}_{\mathtt{G}}$ – to $\mathtt{P}|_{\mathcal{E}_{\mathtt{G}}}$ in all possible ways. Therefore we can conclude $\mathcal{S}(\mathtt{G}') \subseteq \mathtt{sat}(\mathcal{S}(\mathtt{G}), \mathtt{f})$. Note that, if $\mathtt{f}'$ is a root ℓ-extension, $\mathbb{M}(\mathcal{S})(\mathtt{f})$ must be an ℓ-extension, as it has the same underlying function between events.

Finally, we need to show that $\mathbb{M}(\mathcal{S})$ weekly preserves colimits. Preservation of the initial object is easy, since $\mathcal{S}(\epsilon) = \{\epsilon\}$ by definition (see Definition 2.1) and the right hand side is the initial object in $\mathbf{SPath}(\mathcal{L})$. As for pushouts: Since $\mathcal{S}$ is coherent, preservation boils down to Lemma 3.26. □

We now give our completeness result.

Theorem 3.33. *Any coherent functor* $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ *induces a coherent specification* $\mathbb{S}(\mathbb{F})$*.*

Proof. The claim follows from Lemma 3.31 by setting $\mathbb{S}(\mathbb{F})(\mathtt{G}) = \mathbb{F}(\mathtt{G})$. □

3.6. More completeness

We can sharpen the completeness result by requiring functors to preserve specific properties for suitable arrows of $\mathbf{PIDag}(\mathcal{L})$. The candidates are root extension morphisms, given the properties shown in Section 3.3. In order to define the functors, we also need to consider a subset of the arrows of $\mathbf{SPath}(\mathcal{L})$.

Definition 3.34 (Saturated specifications)**.** *Let $\mathcal{S}$ be a specification. It is* saturated *if for all graphs $\mathtt{G}$, the function $\mathtt{f} : \mathcal{E}_{\mathtt{G}} \to \mathcal{E}_{\mathtt{G}^{\ell}_{\mathcal{E}}}$ is saturated with respect to $\mathcal{S}(\mathtt{G}^{\ell}_{\mathcal{E}})$ (see Lemma 3.20), that is*

$$\forall \mathtt{G}, \mathcal{E}, \ell.\ \mathcal{S}(\mathtt{G}^{\ell}_{\mathcal{E}}) = \mathtt{sat}(\mathtt{ret}(\mathcal{S}(\mathtt{G}^{\ell}_{\mathcal{E}}), \mathtt{f}), \mathtt{f})$$

A *saturation ps-morphism* (along ℓ) is a ps-extension $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ along ℓ whose underlying function $\mathtt{f} \colon \mathcal{E} \to \mathcal{E} \cup \{\top\}$ is saturated. We can now prove an instance of Theorem 3.33 concerning saturated specifications.

Proposition 3.35. *Let $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ be a coherent functor. If it maps root extension morphisms to saturation ps-morphisms (along the same labels), then the induced specification $\mathbb{S}(\mathbb{F})$ is saturated and coherent.*

Proof. We know by Theorem 3.33 that $\mathbb{S}(\mathbb{F})$ is coherent. It remains to be shown that $\mathbb{S}(\mathbb{F})$ is saturated, that is $\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}) = \mathtt{sat}(\mathtt{ret}(\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}), \mathtt{f}), \mathtt{f})$. If $\mathtt{G}^{\ell}_{\mathcal{E}}$ is rooted, this follows from $\mathbb{F}$ mapping root extensions to saturation ps-morphisms. Otherwise, by coherence, $\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}})$ can be decomposed into the product $\bigotimes_{\mathtt{e} \in (\mathcal{E}_{\mathtt{G}})_{\top}} \mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}|_{[\mathtt{e}]})$. For each component of the product we have a root extension $\mathtt{G}^{\ell}_{\mathcal{E}}|_{[\mathtt{e}]} \setminus \mathtt{e} \to \mathtt{G}^{\ell}_{\mathcal{E}}|_{[\mathtt{e}]}$, which is mapped by $\mathbb{F}$ to a saturation ps-morphism, therefore we have $\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}|_{[\mathtt{e}]}) = \mathtt{sat}(\mathtt{ret}(\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}|_{[\mathtt{e}]}), \mathtt{f}_{\mathtt{e}}), \mathtt{f}_{\mathtt{e}})$, where $\mathtt{f}_{\mathtt{e}}$ is the underlying function between events of the root extension. Saturation of $\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}})$ follows by computing the product of these sets of paths. □

The need of finding a suitable image for root extension morphisms allows for alternative choices. To this end, we introduce a different subset of the arrows of $\mathbf{SPath}(\mathcal{L})$.

Definition 3.36 (Path extension/prefixing)**.** *Let $\mathtt{P}$ be a path and $\mathtt{f} : (\mathcal{E}_{\mathtt{P}}, \lambda_{\mathtt{P}}) \to (\mathcal{E}, \lambda)$ a function preserving labels. The extension of $\mathtt{P}$ along $\mathtt{f}$ is defined as*

$$\mathtt{ext}(\mathtt{P}, \mathtt{f}) = \{\mathtt{Q} \mid \mathtt{Q} \in \mathbb{P}(\mathcal{E}, \lambda) \textit{ and } \mathtt{f} \textit{ induces a pr-morphism } \mathtt{f} : \mathtt{P} \to \mathtt{Q} \textit{ in } \mathbf{Path}(\mathcal{L})\}$$

Similarly, let $\mathtt{Q}$ be a path and $\mathtt{f} : \mathcal{E} \to \mathcal{E}_{\mathtt{Q}}$ a function preserving labels. The prefixing of $\mathtt{Q}$ along $\mathtt{f}$ is defined as

$$\mathtt{pre}(\mathtt{Q}, \mathtt{f}) = \{\mathtt{P} \mid \mathtt{P} \in \mathbb{P}(\mathcal{E}, \lambda) \textit{ and } \mathtt{f} \textit{ induces a pr-morphism } \mathtt{f} : \mathtt{P} \to \mathtt{Q} \textit{ in } \mathbf{Path}(\mathcal{L})\}$$

Both definitions immediately extend to sets of paths. Should $\mathtt{f}$ be injective, $\mathtt{pre}(\mathtt{Q}, \mathtt{f})$ would be a singleton, and if $\mathtt{f}$ is an inclusion, then $\mathtt{pre}(\mathtt{Q}, \mathtt{f}) = \mathtt{Q}|_{\mathcal{E}}$, for the latter a prefix of $\mathtt{Q}$. Also, note that similarly $\mathtt{P}$ has to be a prefix for all the paths in $\mathtt{ext}(\mathtt{P}, \mathtt{f})$.

Example 3.37. *A topological specification* $\mathcal{S}_{topR}$ *for a* `Register` *can be defined as* $\mathcal{S}_{lwwR}$ *in example 2.2 with the additional requirement that paths are topological orderings of visibilities*

$$\mathtt{P} \in \mathcal{S}_{topR}(\mathtt{G}) \quad \textit{iff}\ \mathtt{P} \in \mathcal{S}_{lwwR}(\mathtt{G}) \ \textit{and}\ \prec_{\mathtt{G}} \subseteq \leq_{\mathtt{P}}$$

In this way, $\mathcal{S}_{topR}(\mathtt{G})$ *excludes e.g. the two right-most arbitrations of the equation in Figure 1a.*

Definition 3.38 (Topological specifications)**.** *Let* $\mathcal{S}$ *be a specification. It is* topological *if*

$$\forall \mathtt{G}, \mathcal{E}, \ell.\ \mathcal{S}(\mathtt{G}^{\ell}_{\mathcal{E}}) = \mathtt{ext}(\mathtt{pre}(\mathcal{S}(\mathtt{G}^{\ell}_{\mathcal{E}}), \mathtt{f}), \mathtt{f})$$

A topological ps-morphism *(along* ℓ*) is a ps-extension* $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ *with underlying function* $\mathtt{f} \colon \mathcal{E} \to \mathcal{E} \cup \{\top\}$ *such that* $\mathcal{X}_2 = \mathtt{ext}(\mathtt{pre}(\mathcal{S}(\mathcal{X}_2), \mathtt{f}), \mathtt{f})$.

The name is directly reminiscent of what are called topological RDTs in [14, 8], and in fact it similarly guarantees that arbitrations preserve the visibility order. We can thus prove another instance of Theorem 3.33, now concerning topological specifications.

Proposition 3.39. *Let* $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ *be a coherent functor. If it maps root extension morphisms into topological ps-morphisms (along the same labels), then the induced specification* $\mathbb{S}(\mathbb{F})$ *is coherent and topological.*

3.7. Interchangeability of Functors and Coherent Specifications

The connection between the constructions of Theorem 3.32 and Theorem 3.33 is quite tight, and in fact induces a one-to-one correspondence between functors and coherent specifications.

Theorem 3.40. *Let* $\mathcal{S}$ *be a coherent specification. Then* $\mathbb{S}(\mathbb{M}(\mathcal{S})) = \mathcal{S}$*. Conversely, let* $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ *be a coherent functor. Then* $\mathbb{M}(\mathbb{S}(\mathbb{F})) \simeq \mathbb{F}$.

Proof. We first show that $\mathbb{M}(\mathbb{S}(\mathbb{F})) \simeq \mathbb{F}$. For notational convenience, we denote $\mathbb{M}(\mathbb{S}(\mathbb{F}))$ by $\mathbb{M}'$. We will show the existence of a natural isomorphism $\varphi \colon \mathbb{M}' \Rightarrow \mathbb{F}$. By definition, we have $\mathbb{M}'(\mathtt{G}) = \mathbb{S}(\mathbb{F})(\mathtt{G}) = \mathbb{F}(\mathtt{G})$, therefore we can define $\varphi_{\mathtt{G}} = \textsc{id}_{\mathbb{F}(\mathtt{G})}$. We need to prove that it is natural, which in this case amounts to showing $\mathbb{M}'(\mathtt{f}) = \mathbb{F}(\mathtt{f})$ for $\mathtt{f} \colon \mathtt{G} \to \mathtt{G}'$ in $\mathbf{PIDag}(\mathcal{L})$. This follows from $\mathbb{M}'(\mathtt{f})$ and $\mathbb{F}(\mathtt{f})$ having the same underlying function between events, namely the inclusion $(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}}) \to (\mathcal{E}_{\mathtt{G}'}, \lambda_{\mathtt{G}'})$.

Now we show that $\mathbb{S}(\mathbb{M}(\mathcal{S})) = \mathcal{S}$ for any coherent specification $\mathcal{S}$. This follows directly from the definition of $\mathbb{M}$ and $\mathbb{S}$. In fact, $\mathbb{S}(\mathbb{M}(\mathcal{S}))(\mathtt{G}) = \mathbb{M}(\mathcal{S})(\mathtt{G}) = \mathcal{S}(\mathtt{G})$. □

The one-to-one correspondence can be lifted to the specific classes of saturated and topological coherent specifications and to the functors of Proposition 3.35 and Proposition 3.39, respectively. However, what is most relevant is the fact the interchangeability allows one to leverage the categorical machinery of the functor category for providing operators on specifications.

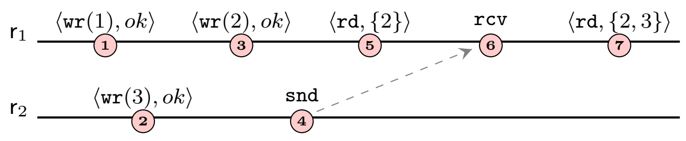
$\mathtt{r}_1$
$\langle \mathtt{wr}(1), ok\rangle$
$\langle \mathtt{wr}(2), ok\rangle$
$\langle \mathtt{rd}, \{2\}\rangle$
rcv
$\langle \mathtt{rd}, \{2,3\}\rangle$
1
3
5
6
7
$\mathtt{r}_2$
$\langle \mathtt{wr}(3), ok\rangle$
snd
2
4

Figure 5: Execution

Remark 3.41. *Besides coherence, one of the keys of the previous correspondence is the (quite reasonable) choice of specifications that preserve isomorphisms. In general terms, whenever one needs to consider the relationship between different specifications, it is necessary to take into account how the underlying sets of events are related. This is quite easy to accomplish if we move to the functorial presentation. For example, we can say that a specification $\mathcal{S}_1$ refines a specification $\mathcal{S}_2$ if $\mathcal{S}_1(\mathtt{G}) \subseteq \mathcal{S}_2(\mathtt{G})$ for all graphs $\mathtt{G}$. However, this is a very concrete characterisation: It would be more general to check for the existence of a ps-morphism $\mathcal{S}_2(\mathtt{G}_2) \to \mathcal{S}_1(\mathtt{G}_1)$ whose underlying function $\mathtt{f} : \mathcal{E}_{\mathtt{G}_2} \to \mathcal{E}_{\mathtt{G}_1}$ is a bijection, in order to abstract from the identities of the events. In this case, a further constraint would be that $\mathtt{f}$ is preserved along the image of the morphisms in $\mathbf{PIDag}(\mathcal{L})$. These requirements boil down to the existence of a natural transformation $\mathbb{M}(\mathcal{S}_2) \to \mathbb{M}(\mathcal{S}_1)$.*

4. On the implementations of Replicated Data Types

An RDT is implemented on top of a set of replicas, which serve requests from clients according to their local state and communicate asynchronously their local changes. Fig. 5 illustrates a scenario involving two replicas, namely $\mathtt{r}_1$ and $\mathtt{r}_2$, that implement a *multi-value Register* (as specified in Example 2.4). A horizontal line corresponds to a replica and shows the relative order (from left to right) in which events occur in that replica. The depicted scenario shows a concrete execution that generates the visibility graph in Fig. 3. The two writes on $\mathtt{r}_1$ are totally ordered (events **1** and **3**), and consequently, 2 overwrites 1. The remaining write takes place on $\mathtt{r}_2$ (event **2**) and is unknown to $\mathtt{r}_1$ until $\mathtt{r}_2$ propagates its changes. Hence, the first read on $\mathtt{r}_1$ (event **5**) returns 2, which is the last written value in $\mathtt{r}_1$. Replicas communicate their local changes by using primitives `snd` and `rcv`. Event **6** in $\mathtt{r}_1$ denotes the synchronisation of its local state with the state of $\mathtt{r}_2$, i.e., $\mathtt{r}_1$ becomes aware of the written value 3 (depicted by the dashed line between events **4** and **6**). Since writes in $\mathtt{r}_1$ and $\mathtt{r}_2$ are concurrent, the last read on $\mathtt{r}_1$ returns the set of maximal concurrent updates, i.e., $\{2, 3\}$.

A crucial aspect in the implementation of RDTs concerns the information exchanged through `snd` and `rcv`. Under the *state-based* approach, replicas communicate their own local states [7] while they only communicate operations (or their effects) under the *operation-based* approach [20]. Hereafter, we will focus on state-based implementations.

4.1. Implementations as RLTSs

Following the approach in [7], we describe an implementation of a data type in terms of the behaviour of a replica and we assume that all the replicas of an implementation behave the same.

We define the behaviour of a replica implementing a specification as a labelled transition system, which we call *Replica LTS* (RLTS). We write Σ for the set of the possible states $\sigma, \sigma_0, \ldots$ of a replica. We assume that Σ forms a *commutative monoid*, namely that it can be equipped with an operation $\oplus$ describing how one replica updates its state upon reception of the local state of another replica. We deviate from previous proposals, such as [7, 9], and treat operations that may change the state of a replica (i.e., *mutators*) differently from those that can only retrieve information from the state (i.e., *queries*). The foremost reason of this distinction is behavioural: The mutators are total, meaning that they can originate from any state; furthermore, they respect the monoidal structure of states. By defining RLTSs over mutators, we will be able to capture the compositional core of RDT implementations. Queries are recovered in our notion of bisimulation for RLTSs; this will be akin to higher-order bisimulation (see, e.g., [16]), where queries are modelled as *barbs*, i.e., predicates on states. Hereafter, we use $\mathcal{M}$ and $\mathcal{B}$ to respectively denote the sets of labels associated with mutators and queries.

Definition 4.1 (Replica LTS)**.** *A* replica LTS *(RLTS in short) on a set of actions $\mathcal{M}$ is a quadruple $\mathcal{Q} = \langle \Sigma, \oplus, 1, \rightsquigarrow \rangle$ such that $\langle \Sigma, \oplus, 1 \rangle$ is a (commutative) monoid and $\rightsquigarrow \subseteq \Sigma \times (\mathcal{M} \cup \Sigma) \times \Sigma$ is a transition relation satisfying*

$$(\textsc{rcv})\ \frac{}{\sigma \overset{\sigma'}{\rightsquigarrow} \sigma \oplus \sigma'}$$

It is action-deterministic *if the relation $\rightsquigarrow_{\mathcal{M}} = \rightsquigarrow \cap (\Sigma \times \mathcal{M} \times \Sigma)$ is a function $(\Sigma \times \mathcal{M}) \rightsquigarrow \Sigma$, and it is* state-deterministic *if the relation $\rightsquigarrow_{\Sigma} = \rightsquigarrow \cap (\Sigma \times \Sigma \times \Sigma)$ is a function $(\Sigma \times \Sigma) \rightsquigarrow \Sigma$.*

According to the rule (RCV), when receiving a state σ' from other replica, the local state σ is updated by combining the information in σ and σ'. Thus, the concrete LTS defining the behaviour of a replica that implements an RDT is obtained by instantiating the definitions of state and state combination.

Remark 4.2. *Note that for state-deterministic RLTs the relation $\rightsquigarrow$ coincides with the closure of $\rightsquigarrow_{\mathcal{M}}$ with respect to the (*RCV*) rule.*

Indeed, all of our examples will be state-deterministic, albeit not action-deterministic. Notably, they will preserve the monoidal operator, i.e., the transition relation will be compositional with respect to state combination.

Definition 4.3 (monoidal RLTS)**.** *A* monoidal *RLTS $\mathcal{Q} = \langle \Sigma, \oplus, 1, \rightsquigarrow \rangle$ is a state-deterministic RLTS such that the transition relation satisfies*

1 if $\sigma_1 \overset{m}{\rightsquigarrow} \sigma_1'$ and $\sigma_2 \overset{m}{\rightsquigarrow} \sigma_2'$ then $\sigma_1 \oplus \sigma_2 \overset{m}{\rightsquigarrow} \sigma_1' \oplus \sigma_2'$

2 if $\sigma_1 \oplus \sigma_2 \overset{m}{\rightsquigarrow} \sigma$ *then* $\exists \sigma_1', \sigma_2'.\ \sigma_1 \overset{m}{\rightsquigarrow} \sigma_1'$, $\sigma_2 \overset{m}{\rightsquigarrow} \sigma_2'$, *and* $\sigma = \sigma_1' \oplus \sigma_2'$

Remark 4.4. *In other terms, in our RLTS we say for a given state how it may a) evolve towards a different state and b) be combined with other states. This is thus more general than the lattice of states (thus with union as state composition) proposed in [20]. First of all, we implicitly consider a pre-order* $\rightsquigarrow$ *on states instead of a partial order, and furthermore we do not require the state composition operator* $\otimes$ *to be induced by the order relation on states. This weakening results in an algebraic structure that allows for modelling a large family of* RDT*s, as shown in Section 4.3.*

We can now move to denote the category of our RLTSs.

Definition 4.5 (category of RLTSs)**.** *The category* **RLTS** *has RLTSs as objects and transition- and monoid-preserving functions as arrows, that is,* $f : \mathcal{Q}_1 \to \mathcal{Q}_2$ *is a monoid homomorphism* $f : \langle \Sigma_1, \oplus_1 \rangle \to \langle \Sigma_2, \oplus_2 \rangle$ *such that* $f(\rightsquigarrow_1) \subseteq \rightsquigarrow_2$.

Remark 4.6. *Note that if* $\mathcal{Q}_2$ *is state-deterministic, then any transition-preserving function* $f : \Sigma_1 \to \Sigma_2$ *is bound to be a monoid homomorphism.*

We finally define an implementation over a family $\mathcal{R}$ of replicas.

Definition 4.7 (RLTS implementation)**.** *An RLTS implementation over* $\mathcal{R}$ *is an* $\mathcal{R}$*-indexed family of isomorphic RLTSs.*

4.2. Bisimulation for RLTSs

We now move to provide a notion of bisimulation for RLTSs, which is akin to the one for higher-order calculi: beside the symmetric simulation game, it exploits the notion of *barbs*, that is, predicates on states. Introduced in [19] and widely used for the definition of equivalences in process calculus, barbs intuitively express the satisfiability of a observable property by a state. More specifically, we will write $\sigma \downarrow b$ to say that a state σ verifies a property b.

Definition 4.8 (Action and implementation bisimulation)**.** *Let* $\mathcal{Q}$ *be an RLTS over a set of actions* $\mathcal{M}$*,* $\mathcal{B}$ *a set of barbs, and* $\downarrow \subseteq \Sigma \times \mathcal{B}$ *a relation. Then an action bisimulation is a symmetric relation* $\mathcal{R} \subseteq Q \times Q$ *between states in* $\mathcal{Q}$ *such that if* $(\sigma_1, \sigma_2) \in \mathcal{R}$ *then*

1 if $\sigma_1 \downarrow b$ *then* $\sigma_2 \downarrow b$

2 if $\sigma_1 \overset{m}{\rightsquigarrow} \sigma_1'$ *then* $\exists \sigma_2'\ \sigma_2 \overset{m}{\rightsquigarrow} \sigma_2'$

It is an implementation bisimulation if additionally

3 if $\sigma_1 \overset{\sigma_1'}{\rightsquigarrow} \sigma_1''$ *then* $\exists \sigma_2', \sigma_2''$ *such that* $\sigma_2 \overset{\sigma_2'}{\rightsquigarrow} \sigma_2''$ *and* $\{(\sigma_1', \sigma_2'), (\sigma_1'', \sigma_2'')\} \subseteq \mathcal{R}$.

We write $\sim_{\mathcal{Q}}^{\mathcal{B}}$ *for the largest action bisimulation, and* $\approx_{\mathcal{Q}}^{\mathcal{B}}$ *for the largest implementation bisimulation.*

Clearly, $\approx^{\mathcal{B}}_{\mathcal{Q}} \subseteq \sim^{\mathcal{B}}_{\mathcal{Q}}$, that is, if $\sigma_1 \approx^{\mathcal{B}}_{\mathcal{Q}} \sigma_2$ then $\sigma_1 \sim^{\mathcal{B}}_{\mathcal{Q}} \sigma_2$ for any two states σ_1, σ_2. However, under certain conditions the two notions coincide.

Proposition 4.9 (Actions as implementations)**.** *Let $\mathcal{Q}$ be a state-deterministic RLTS and $\mathcal{B}$ a set of barbs. Then $\sim^{\mathcal{B}}_{\mathcal{Q}}$ and $\approx^{\mathcal{B}}_{\mathcal{Q}}$ coincide if and only if*

$$\forall \sigma_1, \sigma_2.\ \langle \sigma_1, \sigma_2 \rangle \in \sim^{\mathcal{B}}_{\mathcal{Q}} \implies \forall \sigma'_1.\ \exists \sigma'_2.\ \{\langle \sigma'_1, \sigma'_2 \rangle, \langle \sigma_1 \otimes \sigma'_1, \sigma_2 \otimes \sigma'_2 \rangle\} \subseteq \sim^{\mathcal{B}}_{\mathcal{Q}}$$

Proof. For the left-to-right part, assume $\sim^{\mathcal{B}}_{\mathcal{Q}} = \approx^{\mathcal{B}}_{\mathcal{Q}}$ and consider $\langle \sigma_1, \sigma_2 \rangle \in \sim^{\mathcal{B}}_{\mathcal{Q}}$. By state-determinism, for all σ'_1 there is a transition

$$\sigma_1 \overset{\sigma'_1}{\rightsquigarrow} \sigma_1 \otimes \sigma'_1.$$

By $\sim^{\mathcal{B}}_{\mathcal{Q}} = \approx^{\mathcal{B}}_{\mathcal{Q}}$ and the definition of implementation bisimulation, this transition can be simulated by $\sigma_2 \overset{\sigma'_2}{\rightsquigarrow} \sigma_1 \otimes \sigma'_2$, for some σ'_2, with

$$\{\langle \sigma'_1, \sigma'_2 \rangle, \langle \sigma_1 \otimes \sigma'_1, \sigma_2 \otimes \sigma'_2 \rangle\} \subseteq \sim^{\mathcal{B}}_{\mathcal{Q}}$$

as required.

Vice versa, suppose the implication holds; we need to prove $\sim^{\mathcal{B}}_{\mathcal{Q}} \subseteq \approx^{\mathcal{B}}_{\mathcal{Q}}$. Consider $\langle \sigma_1, \sigma_2 \rangle \in \sim^{\mathcal{B}}_{\mathcal{Q}}$, and suppose $\sigma_1 \overset{\sigma'_1}{\rightsquigarrow} \sigma_1 \otimes \sigma'_1$. Then by the implication we get the implementation bisimulation condition. □

Note that the property required in Proposition 4.9 for the correspondence between the two equivalences is weaker than the closure with respect to the monoidal operator, which can be stated as

$$\forall \sigma_1, \sigma_2.\ \langle \sigma_1, \sigma_2 \rangle \in \sim^{\mathcal{B}}_{\mathcal{Q}} \implies \forall \sigma.\ \langle \sigma_1 \otimes \sigma, \sigma_2 \otimes \sigma \rangle \in \sim^{\mathcal{B}}_{\mathcal{Q}}$$

4.3. Some key examples

We now present instantiations corresponding to well-known implementations of the RDTs specified in Section 2.2. All the associated RLTSs are state-deterministic, hence we will drop the adjectives. All the RLTSs will be proved to be monoidal, which validates the generality of our model.

Example 4.10 (Implementation of the *GCounter*)**.** *The RLTS corresponding to a replica* r *that implements the* RDT GCounter *specified in Example 2.2 is defined as* $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}} \rangle$ *where*

- $\Sigma = \mathbb{N}^{\mathcal{R}}$*, i.e., states are mappings* v *where* $v(\mathsf{r})$ *is the number of increments performed over* r*.*
- *The operator* $\oplus$ *is defined as the point-wise maximum of* v *and* v', $v \oplus v' = \max\{v, v'\}$ *where* $\max\{v, v'\}(\mathsf{s}) = \max\{v(\mathsf{s}), v'(\mathsf{s})\}$ *for all* $\mathsf{s} \in \mathcal{R}$*.*
- 1 *is the constant map to* 0*.*

- *The set of mutators is the singleton $\mathcal{M} = \{\langle \mathtt{inc}, ok\rangle\}$. Then, $\rightsquigarrow^{\mathsf{r}}$ is defined as the least relation satisfying rule* (RCV) *and the following one*

$$\frac{}{v \overset{\langle \mathtt{inc}, ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} v[\mathsf{r} \mapsto v(\mathsf{r}) + 1]}\ (\text{INC})$$

  which describes the effects of performing an increment over the replica r*: The entry associated with the replica* r *in the map v is incremented.*

It straightforwardly follows from the definitions above that $\langle \Sigma, \oplus, 1\rangle$ is a commutative monoid. We now inspect the conditions in Definition 4.3

1 if $v_i \overset{\langle \mathtt{inc}, ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} v'_i$ then $v'_i = v_i[\mathsf{r} \mapsto v_i(\mathsf{r}) + 1]$ for $i = 1, 2$. Then,

$$\begin{aligned} v'_1 \oplus v'_2 &= (v_1 \oplus v_2)[\mathsf{r} \mapsto \max\{v_1(\mathsf{r}) + 1, v_2(\mathsf{r}) + 1\}] \\ &= (v_1 \oplus v_2)[\mathsf{r} \mapsto \max\{v_1(\mathsf{r}), v_2(\mathsf{r})\} + 1] \\ &= (v_1 \oplus v_2)[\mathsf{r} \mapsto (v_1 \oplus v_2)(\mathsf{r}) + 1] \end{aligned}$$

Hence, $v_1 \oplus v_2 \overset{\langle \mathtt{inc}, ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} v'_1 \oplus v'_2$.

2 if $v_1 \oplus v_2 \overset{\langle \mathtt{inc}, ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} v$ then $v = (v_1 \oplus v_2)[\mathsf{r} \mapsto (v_1 \oplus v_2)(\mathsf{r}) + 1]$. Moreover, $v_i \overset{\langle \mathtt{inc}, ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} [\mathsf{r} \mapsto v_i(\mathsf{r}) + 1]$ for $i = 1, 2$. Hence, the condition follows by taking $v'_i = v_i[\mathsf{r} \mapsto v_i(\mathsf{r}) + 1]$.

Consequently, $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}}\rangle$ is a monoidal RLTS.
The set of barbs is $\mathcal{B} = \{\langle \mathtt{rd}, k\rangle \mid k \in \mathbb{N}\}$, and $\downarrow$ is given by

$$v \downarrow \langle \mathtt{rd}, k\rangle \iff k = \Sigma_{\mathsf{s} \in \mathcal{R}} v(\mathsf{s})$$

Note that $v \downarrow \langle \mathtt{rd}, k\rangle$ holds only if the return value k in the barb corresponds to the value stored in v (i.e., $k = \Sigma_{\mathsf{s} \in \mathcal{R}} v(\mathsf{s})$).

Example 4.11 (*GCounter* bisimulation)**.** *Consider a* GCounter *implementation with two replicas, so that states can be simply modelled as pairs of natural numbers $\langle n, m\rangle$. Also, let us consider the RLTS for the first replica, so that $\langle n, m\rangle \overset{\langle \mathtt{inc}, ok\rangle}{\rightsquigarrow}{}^{1} \langle n + 1, m\rangle$.*

Clearly we have that $\langle n, m\rangle \sim^{\mathcal{B}}_{\mathcal{Q}} \langle x, y\rangle$ if and only if $n + m = x + y$, since $\langle n, m\rangle \downarrow \langle \mathtt{rd}, k\rangle \iff n + m = k$. However, this is not an implementation bisimulation. In fact, let us consider the states $\langle 9, 2\rangle$ and $\langle 6, 5\rangle$: If the former receives the state $\langle 0, 3\rangle$, there is no state that can be suitably received from $\langle 6, 5\rangle$.

Example 4.12 (Implementation of the PN-Counter)**.** *The RLTS for the implementation of the replica r of the* RDT PNCounter *specified in Example 2.3 can be defined as $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}}\rangle$ where*

- $\Sigma = \mathbb{N}^{\mathcal{R}} \times \mathbb{N}^{\mathcal{R}}$, *i.e., states are pairs* (p, n), *where the mappings* p *and* n *respectively record the number of increments and decrements performed over each replica.*
- *The combinator* $\oplus$ *is defined as* $(p, n) \oplus (p', n') = (\max\{p, p'\}, \max\{n, n'\})$.
- *The identity* 1 *is the pair of constant mappings to* 0.
- *The set of mutators is* $\mathcal{M} = \{\langle \mathtt{inc}, ok\rangle, \langle \mathtt{dec}, ok\rangle\}$. *Then,* $\rightsquigarrow^{\mathsf{r}}$ *is defined as the least relation satisfying rule* (RCV) *and the following ones*

$$\frac{}{(p, n) \overset{\langle \mathtt{inc}, ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} (p[\mathsf{r} \mapsto p(\mathsf{r}) + 1], n)}\ (\textsc{inc})$$

$$\frac{}{(p, n) \overset{\langle \mathtt{dec}, ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} (p, n[\mathsf{r} \mapsto p(\mathsf{r}) + 1])}\ (\textsc{dec})$$

As for the GCounter*, we can conclude that* $\langle \Sigma, \oplus, 1\rangle$ *is a monoidal RLTS. Also, the set of barbs is defined as* $\mathcal{B} = \{\langle \mathtt{rd}, k\rangle \mid k \in \mathbb{N}\}$, *and* $\downarrow$ *is given by*

$$(p, n) \downarrow \langle \mathtt{rd}, k\rangle \iff k = \Sigma_{\mathsf{s} \in \mathcal{R}}\ p(\mathsf{s}) - n(\mathsf{s}).$$

Example 4.13 (Implementation of the Multi-value Register)**.** *We present an implementation for the* Multi-value Register *specified in Example 2.4 based on the optimised implementation proposed in [24]. The implementation associates a single scalar (logical) clock and a vector clock to each written value to determine if two writes are concurrent or causally ordered. A scalar clock is a natural number and a version vector is just a mapping from replicas to natural numbers. The state of a replica consists of a pair* (S, v) *where* $S \in 2^{\mathcal{R} \times \mathbb{N} \times \mathcal{V}}$ *is a set of tagged values and* $v \in \mathbb{N}^{\mathcal{R}}$ *a version vector. An element* $(\mathsf{r}, c, k) \in S$ *indicates that the value* k *has been written over* r *at the logical time* c. *It is assumed that* $(\mathcal{V}, \leq)$ *is a partial order. Then, the order among tagged elements is given by*

$$(\mathsf{r}, c, k) \leq (\mathsf{r}', c', k') \iff (\mathsf{r} = \mathsf{r}' \land\ (c, k) \leq_{lex} (c', k')) \lor (\mathsf{r} \neq \mathsf{r}' \land k \leq k')$$

where $\leq_{lex}$ *stands for lexicographical order. Consequently, the maximal concurrent written values in a set* $S \in 2^{\mathcal{R} \times \mathbb{N} \times \mathcal{V}}$ *are the* maximal elements $\|S\|$ *defined as* $\{u \in S \mid \nexists v \in S, u \leq v\}$.

Then, for definition of $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}}\rangle$ *we take*

- $\Sigma = \{(\|S\|, v) \mid S \in 2^{\mathcal{R} \times \mathbb{N} \times \mathcal{V}}, v \in \mathbb{N}^{\mathcal{R}}, \forall (\mathsf{r}, c, k) \in S.1 \leq c \leq v(\mathsf{r})\}$, *i.e., each state consists of the set of maximal concurrent written values tagged with a logical clock consistent with the version vector* v.
- *The state combinator* $\oplus$ *is defined as follows*

$$(S, v) \oplus (S', v') = (\|(S \cap S') \cup S_{|v'} \cup S'_{|v}\|, \max\{v, v'\})$$

where $S_{|v} = \{(\mathsf{r}, c, k) \mid (\mathsf{r}, c, k) \in S\ \land c > v(\mathsf{r})\}$

- *the identity* 1 *is the pair* $(\emptyset, \bot)$ *where* $\bot$ *is the constant mapping to* 0.
- *The set of mutators is* $\mathcal{M} = \{\langle \mathtt{wr}(k), ok\rangle \mid k \in \mathcal{V}\}$. *Then,* $\rightsquigarrow^{\mathsf{r}}$ *is defined as the least relation satisfying rule* (RCV) *and the following one*

$$\frac{}{(S, v)\overset{\langle \mathtt{wr}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}}(\{(\mathsf{r}, v(\mathsf{r})+1, k)\}, v[\mathsf{r} \mapsto v(\mathsf{r})+1])}\ (\text{WR})$$

It is immediate from its definition that $\oplus$ *is commutative. We show that* $1 = (\emptyset, \bot)$ *is the identity as follows.*

$$\begin{array}{rll}(S,v)\oplus(\emptyset,\bot) & = (\|(S\cap\emptyset)\cup S_{|\bot}\cup\emptyset_{|v}\|, \max\{v,\bot\}) & \textit{by the def. of } \oplus\\ & = (\|S_{|\bot}\cup\emptyset_{|v}\|, \max\{v,\bot\}) & \textit{by the def. of } \cup \textit{ and } \cap\\ & = (\|S_{|\bot}\|, \max\{v,\bot\}) & \textit{because } \emptyset_{|v}=\emptyset\\ & = (\|S_{|\bot}\|, v) & \textit{because } \bot \textit{ is the identity}\end{array}$$

It remains to show that $\|S_{|\bot}\| = S$. *Since* $(S, v) \in \Sigma$, *for all* $\forall(\mathsf{r}, c, k) \in S.1 \le c \le v(\mathsf{r})$. *Therefore,* $\forall(\mathsf{r}, c, k) \in S$, $c > 0 = \bot(\mathsf{r})$ *holds. Hence,* $S_{|\bot} = S$. *Moreover,* $(S, v) \in \Sigma$ *implies that there exists* S' *such that* $S = \|S'\|$. *Therefore,* $\|S\| = S$ *because* $\|\_\|$ *is idempotent. Hence, we can conclude that* $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}}\rangle$ *is a commutative monoid. We now inspect the conditions in Definition 4.3*

1 *if* $(S_i, v_i)\overset{\langle \mathtt{wr}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}}(S'_i, v'_i)$ *then we have* $S'_i = \{(k, \mathsf{r}, v(\mathsf{r}) + 1)\}$ *and* $v'_i = v_i[\mathsf{r} \mapsto v(\mathsf{r}) + 1]$ *for* $i = 1, 2$. *Then*

$$(S'_1, v'_1) \oplus (S'_2, v'_2) = (\|(S'_1 \cap S'_2) \cup S'_{1|v'_2} \cup S'_{2|v'_1}\|, \max\{v'_1, v'_2\})$$

Firstly, we note that $\max\{v'_1, v'_2\} = \max\{v_1, v_2\}[\mathsf{r} \mapsto \max\{v_1, v_2\}(\mathsf{r}) + 1]$. *For* $\|(S'_1 \cap S'_2) \cup S'_{1|v'_2} \cup S'_{2|v'_1}\|$, *we have three cases*

- $v_1(\mathsf{r}) = v_2(\mathsf{r})$. *Then* $S'_1 = S'_2 = \{(\mathsf{r}, v_1(\mathsf{r})+1, k)\} = S'_1 \cap S'_2$. *Moreover* $S'_{1|v'_2} = S'_{2|v'_1} = \emptyset$ *because* $S'_i = (\mathsf{r}, v_1(\mathsf{r})+1, k)$ *and clearly* $v_1(\mathsf{r})+1 \not> v'_i(\mathsf{r}) = v_i(\mathsf{r}) + 1$. *Hence*

$$(S_1, v'_1) \oplus (S'_2, v'_2) = (\{(\mathsf{r}, v_1(\mathsf{r}) + 1, k)\}, \max\{v'_1, v'_2\})$$

Therefore $(S_1, v_1) \oplus (S_2, v_2)\overset{\langle \mathtt{wr}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}}(S'_1, v'_1) \oplus (S'_2, v'_2)$.

- $v_1(\mathsf{r}) > v_2(\mathsf{r})$. *Then* $S'_i = \{(\mathsf{r}, v_i(\mathsf{r})+1, k)\}$ *and* $S'_1 \cap S'_2 = \emptyset$. *Moreover* $S'_{1|v'_2} = \{(\mathsf{r}, v_i(\mathsf{r}) + 1, k)\}$ *and* $S'_{2|v'_1} = \emptyset$. *Hence*

$$(S_1, v'_1) \oplus (S'_2, v'_2) = (\{(\mathsf{r}, v_1(\mathsf{r}) + 1, k)\}, \max\{v'_1, v'_2\})$$

Therefore $(S_1, v_1) \oplus (S_2, v_2)\overset{\langle \mathtt{wr}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}}(S'_1, v'_1) \oplus (S'_2, v'_2)$.

- $v_2(\mathsf{r}) > v_1(\mathsf{r})$. *Follows as in the previous case.*

2 Follows along the lines of the previous case.

The set of barbs is defined as $\mathcal{B} = \{\langle \mathtt{rd}, S\rangle_{\mathsf{r}} \mid S \in 2^{\mathcal{V}}\}$*, and* $\downarrow$ *is given by*

$$(S, v) \downarrow \langle \mathtt{rd}, S'\rangle_{\mathsf{r}} \iff S' = \{k \mid (\mathsf{s}, c, k) \in S\}$$

Example 4.14 (Implementation of the Last-write wins Register)**.** *We now describe the implementation based on timestamps proposed in [21] for the* Last-write wins Register *introduced in Example 2.5. Let* $(\mathbb{T}, <)$ *be the totally-ordered set of timestamps with minimum* t_0*. Then, the corresponding RLTS for a replica* r *is defined as* $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}}\rangle$ *where*

- $\Sigma = \mathbb{T} \times (\mathcal{V} \cup \{\bot\})$*, i.e., the state* (t, k) *of a replica contains the current value* k *of the register and its associated timestamp. We assume* $(\mathcal{V}, \leq)$ *to be a total order and establish that* $\bot \leq k$ *for all* $k \in \mathcal{V}$*; moreover, we consider* $\mathbb{T} \times (\mathcal{V} \cup \{\bot\})$ *lexicographically ordered, i.e.,* $(t, k) \leq (t', k')$ *when either* $t < t'$ *or* $t = t'$ *and* $k \leq k'$*.*
- *The state combinator* $\oplus$ *is defined as follows*

  $$(t, k) \oplus (t', k') = \max\{(t, k), (t', k')\}$$

- *the identity* 1 *is the pair* $(t_0, \bot)$*.*
- *The set of mutators is* $\mathcal{M} = \{\langle \mathtt{wr}(k), ok\rangle \mid k \in \mathbb{N}\}$*. Then,* $\rightsquigarrow^{\mathsf{r}}$ *is defined as the least relation satisfying rule* (RCV) *and the following one*

  $$\frac{t' \in \mathbb{T}}{(t, k) \overset{\langle \mathtt{wr}(k'), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} \max\{(t, k), (t', k')\}}\ \text{(WR)}$$

  Note that the transition system is non-deterministic, because the target of the arrow depends on the chosen t'*.*

It immediately follows from the definitions that $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}}\rangle$ *is a commutative monoid. We now inspect the conditions in Definition 4.3*

1 if $(t_i, k_i) \overset{\langle \mathtt{wr}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} (t'_i, k'_i)$ *then either* $(t'_i, k'_i) = (t_i, k_i)$ *or* $(t_i, k_i) < (t'_i, k'_i)$ *and* $k'_i = k$*. Then, define* $(t', k') = (t'_1, k'_1) \oplus (t_2, k'_2) = \max\{(t'_1, k'_1) \oplus (t'_2, k'_2)\}$*. Since* $(t_i, k_i) \leq (t', k')$ *for* $i = 1, 2$*, we conclude that* $(t_1, k_1) \oplus (t_1, k_1) \leq (t', k')$ *and therefore* $(t_1, k_1) \oplus (t_1, k_1) \overset{\langle \mathtt{wr}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} (t', k')$*.*

2 Follows along the lines of the previous case.

The set of barbs is defined as $\mathcal{B} = \{\langle \mathtt{rd}, k\rangle \mid k \in \mathbb{N}\}$*, and* $\downarrow$ *is given by*

$$(t, k) \downarrow \langle \mathtt{rd}, k'\rangle \iff k' = k$$

Example 4.15 (Implementation of the *OR-Set*)**.** *The behaviour of a single replica according to the implementation proposed in [2] is given by the RLTS $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}} \rangle$ defined as follows*

- $\Sigma = \{(v, w) \mid v \in \mathbb{N}^{\mathcal{R}}, w \in \mathbb{N}^{\mathcal{V}\times\mathcal{R}}, w(k, \mathsf{r}) \leq v(\mathsf{r})\}$*, i.e., states are pairs* (v, w)*, where* v *is a version vector, and* w *maps each pair* (k, r) *to the most recent version of* r *in which* k *has been added to* r*. If* $w(k, \mathsf{r}) = 0$ *then* k *does not belong to the set according to* r*.*

- *The combination of* (v, w) *and* (v', w') *handles conflicting information in the mappings* w *and* w' *for the same element. There is a conflict between* w *and* w' *for* (k, s) *when one mapping indicates that* k *is present in the replica* s *and the other does not, i.e.,* $w(k, \mathsf{s}) > 0$ *and* $w'(k, \mathsf{s}) = 0$ *or vice versa. Such conflicts are resolved by using the version vectors* v *and* v'*, as formalised by the operation* $+ : (\mathbb{N}^{\mathcal{R}} \times \mathbb{N}^{\mathcal{V}\times\mathcal{R}}) \times (\mathbb{N}^{\mathcal{R}} \times \mathbb{N}^{\mathcal{V}\times\mathcal{R}}) \to \mathbb{N}^{\mathcal{V}\times\mathcal{R}}$*. defined such that*

$$(v, w) + (v', w')(k, \mathsf{s}) = \begin{cases} w(k, \mathsf{s}) & v(\mathsf{s}) > v'(\mathsf{s}) \\ w'(k, \mathsf{s}) & v(\mathsf{s}) < v'(\mathsf{s}) \\ \min\{w(k, \mathsf{s}), w'(k, \mathsf{s})\} & \textit{otherwise} \end{cases}$$

  Then, the state combinator $\oplus$ *is defined as*

$$(v, w) \oplus (v', w') = (\max\{v, v'\}, (v, w) + (v', w'))$$

- *The identity* 1 *is the pair* $(\bot, \bot)$ *where* $\bot$ *are suitable constant maps to* 0*.*

- *The set of mutators is*

$$\mathcal{M} = \{\langle \mathtt{add}(k), ok\rangle \mid k \in \mathcal{V}\} \cup \{\langle \mathtt{rem}(k), ok\rangle \mid k \in \mathcal{V}\}$$

  Then, $\rightsquigarrow^{\mathsf{r}}$ *is defined as the least relation satisfying rule* (RCV) *and the following ones*

$$\frac{}{(v, w) \overset{\langle \mathtt{add}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} (v[\mathsf{r} \mapsto v(\mathsf{r}) + 1], w[(k, \mathsf{r}) \mapsto v(\mathsf{r}) + 1])} \ (\textsc{add})$$

$$\frac{}{(v, w) \overset{\langle \mathtt{rem}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} (v, w[\forall \mathsf{s} \in \mathcal{R}.\ (k, \mathsf{s}) \mapsto 0])} \ (\textsc{rem})$$

  By rule (ADD)*, if* k *is added to* r*, then* r *changes its local state by (i) creating a new version, i.e, the entry* $v(\mathsf{r})$ *is updated to* $v(\mathsf{r}) + 1$*, and (ii) recording that* k *has been added in the newest version of* r*, i.e.* $w(k, \mathsf{r})$ *is updated to* $v(\mathsf{r}) + 1$*. By rule* (REM)*, if* k *is removed from the set, then all entries in the second mapping associated with* k *are set to 0 to record the elimination.*

First, note that + is a commutative operator. Hence, $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}} \rangle$ is a commutative monoid. We now inspect the conditions in Definition 4.3

1 *if* $(v_i, w_i) \overset{\langle \mathtt{add}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} (v_i', w_i')$ *then we have* $v_i' = v_i[\mathsf{r} \mapsto v_i(\mathsf{r}) + 1]$ *and* $w_i' = w_i[(k, \mathsf{r}) \mapsto v_i(\mathsf{r}) + 1]$ *for* $i = 1, 2$.

  Then, define $(v', w') = (v_1', w_1') \oplus (v_2', w_2') = (\max\{v_1', v_2'\}, (v_1', w_1') + (v_2', w_2'))$. *Note that* $\max\{v_1', v_2'\} = \max\{v_1, v_2\}[\mathsf{r} \mapsto \max\{v_1, v_2\}(\mathsf{r}) + 1]$. *Also, for all* $\mathsf{s} \in \mathcal{R}$ *and* $k' \in \mathbb{N}$, *if* $\mathsf{s} \neq \mathsf{r}$ *and* $k' \neq k$ *we have*

$$((v_1', w_1') + (v_2', w_2'))(k', \mathsf{s}) = ((v_1, w_1) + (v_2, w_2))(k', \mathsf{s})$$

  Additionally

$$((v_1', w_1') + (v_2', w_2'))(k, \mathsf{r}) = \max\{v_1(\mathsf{r}), v_2(\mathsf{r})\} + 1$$

  Hence

$$(v_1', w_1') + (v_2', w_2') = ((v_1, w_1) + (v_2, w_2))[(k, \mathsf{r}) \mapsto \max\{v_1(\mathsf{r}), v_2(\mathsf{r})\} + 1]$$

  Consequently, $(v_1, w_1) \oplus (v_2, w_2) \overset{\langle \mathtt{add}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} (v', w')$.

  The case for the mutator $\langle \mathtt{rem}(k), ok\rangle$ *follows analogously.*

2 *Follows along the lines of the previous case.*

The set of barbs is $\mathcal{B} = \{\langle \mathtt{lookup}, S\rangle \mid S \in 2^{\mathcal{V}}\}$, *and* $\downarrow$ *is given by*

$$(v, w) \downarrow \langle \mathtt{lookup}, S\rangle \iff S = \{k \mid \exists \mathsf{s} \in \mathcal{R}.\ w(k, \mathsf{s}) > 0\}$$

When performing a lookup, the replica r *returns the set* S *of all* k *that, according to* w*, have been added to some replica* s *(i.e.,* $w(k, \mathsf{s}) > 0$ *holds)*

Example 4.16 (Implementation of the *2P-Set*)**.** *The behaviour of a single replica according to the implementation proposed in [21] is given by the RLTS* $\mathcal{Q}^{\mathsf{r}} = \langle \Sigma, \oplus, 1, \rightsquigarrow^{\mathsf{r}} \rangle$ *defined as follows*

- $\Sigma = 2^{\mathcal{V}} \times 2^{\mathcal{V}}$, *i.e., states are pairs* (A, R)*, where* A *and* R *are respectively the sets of added and removed elements.*
- *The combination of* (A, R) *and* (A', R') *simply consists in the point-wise union of sets, i.e.,* $(A, R) \oplus (A', R') = (A \cup A', R \cup R')$.
- *The identity* 1 *is the pair* $(\emptyset, \emptyset)$.
- *The set of mutators is*

$$\mathcal{M} = \{\langle \mathtt{add}(k), ok\rangle \mid k \in \mathcal{V}\} \cup \{\langle \mathtt{rem}(k), ok\rangle \mid k \in \mathcal{V}\}$$

Then, $\rightsquigarrow^{\mathsf{r}}$ is defined as the least relation satisfying rule (RCV) *and the following ones*

$$\frac{}{(A,R)\overset{\langle \mathtt{add}(k),ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}}(A\cup\{k\},E)}\ (\text{ADD})$$

$$\frac{}{(A,R)\overset{\langle \mathtt{rem}(k),ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}}(A,E\cup\{k\})}\ (\text{REM})$$

According to (ADD)*, an addition simply extends the set of added elements with k; analogously, a removal extends the set of removed events* (ADD)*.*

The implementation is analogous to the PNCounter; consequently, we can show that $\mathcal{Q}$ is a monoidal RLTS.

The set of barbs is defined as $\mathcal{B} = \{\langle \mathtt{lookup}, S\rangle \mid S \in 2^{\mathcal{V}}\}$, and $\downarrow$ is given by

$$(A,R)\downarrow\langle\mathtt{lookup},S\rangle \iff S = A\setminus R$$

When performing a lookup, the replica $\mathtt{r}$ *returns the elements that have been added but not removed.*

5. From specifications to LTS

We can exploit the structure of coherent functors to recover an operational interpretation of specifications. In the following, we consider a coherent functor $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$. We we will recover the corresponding LTS via a standard categorical construction, namely its *category of elements* $\mathbf{E}(\mathbb{F})$ (see e.g. [18, Pag. 41]) as follows. Given a pr-morphism $\mathtt{f}\colon \mathtt{G} \to \mathtt{G}_1$ and $\mathtt{P} \in \mathbb{F}(\mathtt{G})$, we denote by $\mathbb{F}(\mathtt{f})(\mathtt{P})$ the set of paths in $\mathbb{F}(\mathtt{G}_1)$ that are in the "image" of $\mathtt{P}$ via $\mathbb{F}(\mathtt{f})$, formally specified as $\mathbb{F}(\mathtt{G}_1) \cap \mathtt{sat}(\mathtt{P},\mathtt{f})$.

Definition 5.1 (Category of elements)**.** *The category of elements $\mathbf{E}(\mathbb{F})$ of $\mathbb{F}$ is obtained as follows*

- *objects are pairs $\langle\mathtt{G},\mathtt{P}\rangle$, such that $\mathtt{G} \in \mathbf{PIDag}(\mathcal{L})$ and $\mathtt{P} \in \mathbb{F}(\mathtt{G})$;*
- *arrows $\mathtt{f}\colon \langle\mathtt{G},\mathtt{P}\rangle \to \langle\mathtt{G}_1,\mathtt{P}_1\rangle$ are pr-morphisms $\mathtt{f}\colon \mathtt{G} \to \mathtt{G}_1$ such that $\mathtt{P}_1 \in \mathbb{F}(\mathtt{f})(\mathtt{P})$.*

Intuitively, arrows in $\mathbf{E}(\mathbb{F})$ stand for the possible ways a path in $\mathbb{F}(\mathtt{G})$ can evolve according to $\mathbb{F}(\mathtt{f})$. The category $\mathbf{E}(\mathbb{F})$ is clearly an LTS, since each category is so. We note that our way of distilling an LTS is similar to how one obtains an LTS from a relation presheaf [23, Definition 4.1].

Example 5.2. *Consider the functor $\mathbb{M}(\mathcal{S}_{mvR})$ induced by the coherent specification $\mathcal{S}_{mvR}$ in Example 2.4. An object $\langle\mathtt{G},\mathtt{P}\rangle$ of the category of elements $\mathbf{E}(\mathbb{M}(\mathcal{S}_{mvR}))$ represents a state of the* RDT *where the events in the visibility graph $\mathtt{G}$ are arbitrated according to $\mathtt{P} \in \mathcal{S}_{mvR}(\mathtt{G})$. An arrow $\mathtt{f}\colon \langle\mathtt{G},\mathtt{P}\rangle \to \langle\mathtt{G}_1,\mathtt{P}_1\rangle$ in $\mathbf{E}(\mathbb{M}(\mathcal{S}_{mvR}))$ describes a computation where the visibility $\mathtt{G}$ is extended to $\mathtt{G}_1$ and the arbitration $\mathtt{P}$ to $\mathtt{P}_1$. For instance, take the graphs $\mathtt{G}_2$ and $\mathtt{G}_3$ in Fig. 4c* *and Fig. 4d, and the unique pr-morphism* $\mathtt{f}\colon \mathtt{G}_2 \to \mathtt{G}_3$*. If* $\mathtt{P}_2$ *is a total order of the events in* $\mathtt{G}_2$*, then there is a ps-morphism* $\mathtt{f} : \{\mathtt{P}_2\} \to \mathtt{sat}(\mathtt{P}_2, \mathtt{f})$*. Moreover,* $\mathcal{S}_{mvR}(\mathtt{G}_3) \cap \mathtt{sat}(\mathtt{P}_2, \mathtt{f}) = \mathtt{sat}(\mathtt{P}_2, \mathtt{f})$ *because* $\mathcal{S}_{mvR}$ *imposes no constraint on the admissible arbitrations of a consistent visibility. Therefore, there is a morphism* $\mathtt{f}\colon \langle \mathtt{G}_2, \mathtt{P}_2\rangle \to \langle \mathtt{G}_3, \mathtt{P}_3\rangle$ *for any* $\mathtt{P}_3 \in \mathtt{sat}(\mathtt{f}, \mathtt{P}_2)$ *in* $\mathbf{E}(\mathbb{M}(\mathcal{S}_{mvR}))$*.*

A pr-morphism may not induce an arrow in the category of elements, as $\mathtt{f}\colon \mathtt{G}_2 \to \mathtt{G}_4$ *with* $\mathtt{G}_2$ *from Fig. 4c and* $\mathtt{G}_4$ *its root extension along* $\langle \mathtt{rd}, \{1\}\rangle$*. Indeed,* $\mathcal{S}_{mvR}(\mathtt{G}_4) = \emptyset$*, and hence* $\mathbb{M}(\mathcal{S}_{mvR})(\mathtt{f})(\mathtt{P}) = \emptyset$ *for any* $\mathtt{P} \in \mathcal{S}_{mvR}(\mathtt{G}_2)$*.*

An analogous situation occurs when the specification restricts the allowed arbitrations, as $\mathcal{S}_{lwwR}$ *in Example 2.5. Consider the root extension* $\mathtt{f}\colon \mathtt{G} \to \mathtt{G}_1$ *along* $\langle \mathtt{rd}, \{2\}\rangle$*, with* $\mathtt{G}_1$ *as in Fig. 4a and* $\mathtt{G}$ *is* $\mathtt{G}_1$ *without the event* $\langle \mathtt{rd}, \{2\}\rangle$*. Then we have* $\mathcal{S}_{lwwR}(\mathtt{G}_1) = \{\mathtt{P}_1\}$*, and* $\mathcal{S}_{lwwR}(\mathtt{G})$ *contains two paths:* $\mathtt{P}$*, which keeps the order of writes as in* $\mathtt{P}_1$*, and* $\mathtt{P}'$*, which inverts it. The latter cannot be extended to any path in* $\mathcal{S}_{lwwR}(\mathtt{G}_1)$*, as writes are in the wrong order. In fact, there is no* $\mathtt{f}\colon \langle \mathtt{G}, \mathtt{P}'\rangle \to \langle \mathtt{G}_1, \mathtt{P}_1\rangle$ *in* $\mathbf{E}(\mathbb{M}(\mathcal{S}_{lwwR}))$*. Contrastingly, we have that* $\mathtt{f}\colon \langle \mathtt{G}, \mathtt{P}'\rangle \to \langle \mathtt{G}_1, \mathtt{P}''\rangle$ *is an arrow of* $\mathbf{E}(\mathbb{M}(\mathcal{S}_{mvR}))$ *for any* $\mathtt{P}'' \in \mathtt{sat}(\mathtt{f}, \mathtt{P}')$*, as the order of writes is irrelevant for* $\mathcal{S}_{mvR}$*.*

5.1. One- and Multi-replica LTSs

We now recover one-replica LTSs in our setting (see Definition 2.10), by observing that the way visibility is augmented along a transition can be formalised as a root ℓ-extension. In fact, the one-replica LTS precisely corresponds to a sub-category of $\mathbf{E}(\mathbb{F})$ consisting only of such extensions.

Proposition 5.3. *Let* $\mathbf{E}_{\mathsf{o}}(\mathbb{F})$ *be* $\mathbf{E}(\mathbb{F})$ *restricted to root* ℓ*-extensions, for all* $\ell \in \mathcal{L}$*. Then the one-replica LTS coincides with the LTS for* $\mathbf{E}_{\mathsf{o}}(\mathbb{F})$*.*

This is easily seen: Each root ℓ-extension corresponds uniquely to a ℓ-labelled one-replica transition, and between any two graphs there is at most one root extension, so that also the label is implicitly recovered.

We now go a step further, and recover the multi-replica LTS of Definition 2.12 by exploiting the structure of coherent functors. Recall that, given replica states $\langle \mathtt{G}_i, \mathtt{P}_i\rangle$, $i = 1, 2$, we assume that $\mathtt{G}_1$ and $\mathtt{G}_2$ are compatible when forming the composite state. In our categorical setting, compatibility is formalised as the existence of a span $\mathtt{f}_i : \mathtt{G} \to \mathtt{G}_i$ of mono pr-morphisms such that $\mathcal{E}_{\mathtt{G}} = \mathcal{E}_{\mathtt{G}_1} \cap \mathcal{E}_{\mathtt{G}_2}$ (thus, shared nodes have the same labels and predecessors). Notably, the union $\mathtt{G}_1 \sqcup \mathtt{G}_2$ and the obvious morphisms are the pushout for this span.

Before our main characterisation result, we need two technical lemmas. The first says that certain pushouts in $\mathbf{SPath}(\mathcal{L})$ can be decomposed as pushouts over singleton path sets. The second says that every extension is determined by a root extension along the same label.

Lemma 5.4 (Decomposition)**.** *Consider the following diagrams in* $\mathbf{SPath}(\mathcal{L})$

$$\begin{tikzcd}
\mathcal{X} \arrow[r, hook, "\mathtt{f}_2"] \arrow[d, hook, "\mathtt{f}_1"'] & \mathcal{X}_2 \arrow[d, hook, "\mathtt{f}_3"] \\
\mathcal{X}_1 \arrow[r, hook, "\mathtt{f}_4"'] & \mathcal{X}_1 \otimes \mathcal{X}_2
\end{tikzcd}
\qquad
\begin{tikzcd}
\{\mathtt{P}\} \arrow[r, hook, "\bar{\mathtt{f}}_2"] \arrow[d, hook, "\bar{\mathtt{f}}_1"'] & \{\mathtt{P}_2\} \arrow[d, hook, "\bar{\mathtt{f}}_3"] \\
\{\mathtt{P}_1\} \arrow[r, hook, "\bar{\mathtt{f}}_4"'] & \mathtt{P}_1 \otimes \mathtt{P}_2
\end{tikzcd}$$

If the diagram on the left is a pushout, then for all $\mathtt{P}_1 \in \mathcal{X}_1$ *and* $\mathtt{P}_2 \in \mathcal{X}_2$ *there are pushouts as shown on the right such that* $\mathtt{f}_i$ *and* $\overline{\mathtt{f}}_i$ *have the same underlying function on events.*

Lemma 5.5. *Let* $\mathtt{f} : \mathtt{G} \to \mathtt{G}_1$ *be a pr-morphism in* $\mathbf{PIDag}(\mathcal{L})$*. Then it is an* ℓ*-extension if and only if there exists a pushout in* $\mathbf{PDag}(\mathcal{L})$

$$\begin{tikzcd}
\overline{\mathtt{G}} \arrow[r, hook, "\overline{\mathtt{f}}"] \arrow[d, hook] & \overline{\mathtt{G}}_1 \arrow[d, hook] \\
\mathtt{G} \arrow[r, hook, "\mathtt{f}"'] & \mathtt{G}_1
\end{tikzcd}$$

such that $\overline{\mathtt{f}}$ *is a root* ℓ*-extension.*

We now show that transitions of the multi-replica LTS are those corresponding to ℓ-extensions. Intuitively, an ℓ-extension describes a "local" augmentation of a graph, corresponding to a step of computation of a single replica.

Proposition 5.6. *Let* $\mathbf{E}_{\mathtt{m}}(\mathbb{F})$ *be* $\mathbf{E}(\mathbb{F})$ *restricted to* ℓ*-extensions, for all* $\ell \in \mathcal{L}$*. Then the multi-replica LTS coincides with the LTS for* $\mathbf{E}_{\mathtt{m}}(\mathbb{F})$*.*

Proof. We first show that states in $\mathbf{E}(\mathbb{F})$ – and thus in $\mathbf{E}_{\mathtt{m}}(\mathbb{F})$, as they have the same objects – are of the required form. If $\mathtt{G}_1$ and $\mathtt{G}_2$ are compatible, by Lemma 3.31 we obtain $\mathbb{F}(\mathtt{G}_1 \sqcup \mathtt{G}_2) \cong \mathbb{F}(\mathtt{G}_1) \otimes \mathbb{F}(\mathtt{G}_2)$, from which we have that for any $\langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P}\rangle$, as in Definition 5.1, it holds $\mathtt{P} \in \mathbb{F}(\mathtt{G}_1) \otimes \mathbb{F}(\mathtt{G}_2)$, as required.

In order to show that multi-replica transitions correspond to morphisms of $\mathbf{E}_{\mathtt{m}}(\mathbb{F})$, we will show that these morphisms are generated according to the rule (Comp), i.e., every ℓ-extension in $\mathbf{E}_{\mathtt{m}}(\mathbb{F})$ is generated from a suitable rooted ℓ-extension, and vice versa. We show one direction, the other one is analogous. Let $\overline{\mathtt{f}} \colon \langle \mathtt{G}_1, \mathtt{P}_1\rangle \to \langle \mathtt{G}'_1, \mathtt{P}'_1\rangle$ be a rooted ℓ-extension. For any graph $\mathtt{G}_2$ compatible with $\mathtt{G}_1$ we have the following pushout in $\mathbf{PDag}(\mathcal{L})$

$$\begin{tikzcd}
\mathtt{G}_1 \arrow[r, hook, "\overline{\mathtt{f}}"] \arrow[d, hook] & \mathtt{G}'_1 \arrow[d, hook] \\
\mathtt{G}_1 \sqcup \mathtt{G}_2 \arrow[r, hook, "\mathtt{f}"'] & \mathtt{G}'_1 \sqcup \mathtt{G}_2
\end{tikzcd}$$

(note that $\mathtt{G}'_1 \sqcup \mathtt{G}_1 \sqcup \mathtt{G}_2 \cong \mathtt{G}'_1 \sqcup \mathtt{G}_2$, as $\mathtt{G}_1$ is a sub-graph of $\mathtt{G}'_1$), which implies by Lemma 5.5 that $\mathtt{f}$ is a ℓ-extension. We now have to show that for all $\mathtt{P}' \in \mathtt{P} \otimes \mathtt{P}'_1$ there exist morphisms $\mathtt{f} \colon \langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P}\rangle \to \langle \mathtt{G}'_1 \sqcup \mathtt{G}_2, \mathtt{P}'\rangle$ in $\mathbf{E}_{\mathtt{m}}(\mathbb{F})$. By coherence of $\mathbb{F}$ and Lemma 3.31, we have that the pushout above is mapped by $\mathbb{F}$ to a pushout in $\mathbf{SPath}(\mathcal{L})$ of the form

$$\begin{tikzcd}
\mathbb{F}(\mathtt{G}_1) \arrow[r, hook, "\mathbb{F}(\overline{\mathtt{f}})"] \arrow[d, hook] & \mathbb{F}(\mathtt{G}'_1) \arrow[d, hook] \\
\mathbb{F}(\mathtt{G}_1 \sqcup \mathtt{G}_2) \arrow[r, hook, "\mathbb{F}(\mathtt{f})"'] & \mathbb{F}(\mathtt{G}_1 \sqcup \mathtt{G}_2) \otimes \mathbb{F}(\mathtt{G}'_1)
\end{tikzcd}$$

By Lemma 5.4, this implies the existence of a pushout for the span $\{\mathtt{P}\} \hookleftarrow \{\mathtt{P}_1\} \hookrightarrow \{\mathtt{P}'_1\}$, where the bottom leg is a ps-morphism $\{\mathtt{P}\} \hookrightarrow \mathtt{P} \otimes \mathtt{P}'_1$ with the same underlying function on events as $\mathbb{F}(\mathtt{f})$. Therefore, by definition of ps-morphism and $\mathtt{P} \in \mathbb{F}(\mathtt{G}_1 \sqcup \mathtt{G}_2)$, and by $\mathtt{P}'_1 \in \mathbb{F}(\mathtt{G}'_1)$, we obtain

$$\mathtt{P} \otimes \mathtt{P}'_1 \subseteq \mathtt{sat}(\mathtt{P}, \mathbb{F}(\mathtt{f})) \cap \mathbb{F}(\mathtt{G}_1 \sqcup \mathtt{G}_2) \otimes \mathbb{F}(\mathtt{G}'_1)$$

Noting that the formula on the right coincides with $\mathbb{F}(\mathtt{f})(\mathtt{P})$, we obtain the existence of the desired morphisms in $\mathbf{E}_{\mathtt{m}}(\mathbb{F})$, by Definition 5.1. □

Example 5.7. *Consider once more the category $\mathbf{E}(\mathbb{M}(\mathcal{S}_{mvR}))$ discussed in Example 5.2. By Proposition 5.3, the behaviour of a single replica is characterised by morphisms associated with root extensions. The morphism $\mathtt{f}\colon \langle \mathtt{G}_2, \mathtt{P}_2\rangle \to \langle \mathtt{G}_3, \mathtt{P}_3\rangle$ described in Example refex:catelem corresponds to a one-replica transition with label $\langle \mathtt{rd}, \{2, 3\}\rangle$. In fact, its underlying pr-morphism $\mathtt{f}\colon \mathtt{G}_2 \to \mathtt{G}_3$ is a root extension, accounting for the occurrence of the event $\langle \mathtt{rd}, \{2, 3\}\rangle$ that sees any other event in the configuration; this may happen locally if all events in other replicas have been already propagated. Contrastingly, the extension $\mathtt{f}'\colon \mathtt{P}_1 \to \mathtt{G}_2$ accounts for a new event $\langle \mathtt{wr}(3), ok\rangle$ that is unaware of any other event, and hence, executed in a completely different replica. This corresponds to multi-replica transitions $\mathtt{f}'\colon \langle \mathtt{P}_1, \mathtt{P}\rangle \to \langle \mathtt{G}_2, \mathtt{P}'\rangle$, where $\mathtt{P}'$ arbitrates the additional event $\langle \mathtt{wr}(3), ok\rangle$ anywhere in $\mathtt{P}$. The local one-replica execution originating this transition is obtained via Lemma 5.5: It is $\mathtt{f}\colon \langle \emptyset, \emptyset\rangle \to \langle\langle \mathtt{wr}(3), ok\rangle, \langle \mathtt{wr}(3), ok\rangle\rangle$ (here we write $\langle \mathtt{wr}(3), ok\rangle$ for the one-node graph/path), with the underlying pr-morphism $\emptyset \to \langle \mathtt{wr}(3), ok\rangle$ a root morphism.*

5.2. Context LTS

So far, the category of elements allowed to recast the set-theoretical presentation of one- and multi-replica LTSs. However, its strength is in allowing to obtain a new LTS that is reminiscent of the category of contexts à la Leifer-Milner [17], where arrows represent contexts enabling a transition from the source to the target of the arrow. Here, observations are pairs of an event plus an embedding that records how the resulting local visibility embeds into the global one. These additional observations will be needed for defining a correct notion of simulation.

Definition 5.8. *The context LTS is obtained by taking elements $\langle \mathtt{G}, \mathtt{P}\rangle$ of $\mathbf{E}(\mathbb{F})$ as states, and by labelled transitions triples*

$$\langle \mathtt{G}, \mathtt{P}\rangle \xrightarrow{\langle \mathtt{f}, \overline{\mathtt{f}}\rangle} \langle \mathtt{G}_1, \mathtt{P}_1\rangle$$

such that $\mathtt{f} : \langle \mathtt{G}, \mathtt{P}\rangle \to \langle \mathtt{G}_1, \mathtt{P}_1\rangle$ and $\overline{\mathtt{f}} : \langle \overline{\mathtt{G}}_1, \overline{\mathtt{P}}_1\rangle \to \langle \mathtt{G}_1, \mathtt{P}_1\rangle$ are arrows of $\mathbf{E}(\mathbb{F})$ and there exists a pushout in $\mathbf{PDag}(\mathcal{L})$

$$\begin{tikzcd} \overline{\mathtt{G}} \arrow[r, hook] \arrow[d, hook] & \overline{\mathtt{G}}_1 \arrow[d, hook, "\overline{\mathtt{f}}"] \\ \mathtt{G} \arrow[r, hook, "\mathtt{f}"'] & \mathtt{G}_1 \end{tikzcd}$$

Note that each arrow $\mathtt{f} : \langle \mathtt{G}, \mathtt{P} \rangle \to \langle \mathtt{G}_1, \mathtt{P}_1 \rangle$ of $\mathbf{E}(\mathbb{F})$ induces at least one labelled transition (it suffices to consider for $\overline{\mathtt{f}}$ the identity of $\mathtt{G}_1$), but they could actually be more. In fact, all labels can be obtained constructively, since in $\mathbf{PDag}(\mathcal{L})$ pushouts along monos are also pullbacks, and the arrow $[\mathtt{f}, \overline{\mathtt{f}}] \colon \mathtt{G} + \overline{\mathtt{G}}_1 \to \mathtt{G}_1$ uniquely induced by the coproduct must be epi.

Also note that if we restrict to consider only injections, the pair $\langle \mathtt{f}, \overline{\mathtt{f}} \rangle$ is uniquely characterized by $\langle \overline{\mathtt{G}}_1, \overline{\mathtt{P}}_1 \rangle$. In order to simplify some definitions, in the following we abuse notation and denote as $\langle \overline{\mathtt{G}}_1, \overline{\mathtt{P}}_1 \rangle$ such a label $\langle \mathtt{f}, \overline{\mathtt{f}} \rangle$.

Finally, it is noteworthy that the context LTS includes also the one- and multi-replica, as stated by the result below.

Lemma 5.9. *The LTS for $\mathbf{E}_{\mathtt{o}}(\mathbb{F})$ ($\mathbf{E}_{\mathtt{m}}(\mathbb{F})$) coincides with the restriction of the context LTS to transitions whose labels are pairs $\langle \mathtt{f}, \mathtt{id} \rangle$, where $\mathtt{f}$ is a root extension (an extension, respectively).*

For the former, note that if $\mathtt{f}$ is a root extension, then a mono pr-morphism forming a pushout square has to be an isomorphism. Instead, should $\mathtt{f}$ be an extension, a few alternatives for the second component of the label are available, such as taking for $\overline{\mathtt{G}}_1$ the smallest graph such that $\mathtt{G} \sqcup \overline{\mathtt{G}}_1 = \mathtt{G}_1$. However, the choice is immaterial for our later results on simulation, and further abusing notation we simply denote as ℓ a label $\langle \mathtt{f}, \mathtt{id} \rangle$ such that $\mathtt{f}$ is an ℓ-extension.

Example 5.10. *Consider the multi-replica transition arrow $\mathtt{f}' \colon \langle \mathtt{P}_1, \mathtt{P} \rangle \to \langle \mathtt{G}_2, \mathtt{P}' \rangle$ of Example 5.7. Since we have the arrow $\mathtt{f} \colon \langle \emptyset, \emptyset \rangle \to \langle \langle \mathtt{wr}(3), ok \rangle, \langle \mathtt{wr}(3), ok \rangle \rangle$ in the category of elements, $\mathtt{f}'$ yields the following context LTS transition*

$$\langle \mathtt{P}_1, \mathtt{P} \rangle \xrightarrow{\langle \mathtt{f}', \overline{\mathtt{f}}' \rangle} \langle \mathtt{G}_2, \mathtt{P}' \rangle$$

where $\overline{\mathtt{f}}'$ is the embedding of the one-node graph $\langle \mathtt{wr}(3), ok \rangle$ into $\mathtt{G}_2$. As mentioned, we can just use $\langle \langle \mathtt{wr}(3), ok \rangle, \langle \mathtt{wr}(3), ok \rangle \rangle$ as label, since this is uniquely determined. This label conveys the information about the resulting visibility and arbitration pair for the acting replica.

5.3. Implementation correctness via simulation

We are now ready to characterise implementation correctness as a simulation relation between the context LTS and a RLTS implementation.

Definition 5.11 (Implementation correctness)**.** *Let $\mathcal{S}$ be a specification, $\mathcal{C}_{\mathcal{S}}$ the context LTS, and $\mathcal{I}_{\mathcal{S}} = \{\mathcal{I}_{\mathsf{r}}\}_{\mathsf{r} \in \mathcal{R}}$ the RLTS implementation. An* implementation relation *$\mathcal{R}_{\mathcal{S}} = \{\mathcal{R}_{\mathsf{r}}\}_{\mathsf{r} \in \mathcal{R}}$ is a $\mathcal{R}$-family of relations such that $\mathcal{R}_{\mathsf{r}}$ relates states in $\mathcal{I}_{\mathsf{r}}$ and $\mathcal{C}_{\mathcal{S}}$ and if $(\sigma, \langle \mathtt{G}, \mathtt{P} \rangle) \in \mathcal{R}_{\mathsf{r}}$ then*

1. *if $\sigma \downarrow b$ then $\exists \mathtt{G}', \mathtt{P}'$ such that $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{b} \langle \mathtt{G}', \mathtt{P}' \rangle$ and $(\sigma, \langle \mathtt{G}', \mathtt{P}' \rangle) \in \mathcal{R}_{\mathsf{r}}$;*

2. *if $\sigma \overset{m}{\rightsquigarrow}_{\mathsf{r}} \sigma'$ then $\exists \mathtt{G}', \mathtt{P}'$ such that $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{m} \langle \mathtt{G}', \mathtt{P}' \rangle$ and $(\sigma', \langle \mathtt{G}', \mathtt{P}' \rangle) \in \mathcal{R}_{\mathsf{r}}$;*

3. *if $\sigma \overset{\sigma'}{\rightsquigarrow}_{\mathsf{r}} \sigma''$ then $\exists \mathtt{G}', \mathtt{G}'', \mathtt{P}', \mathtt{P}''$ such that $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\langle \mathtt{G}', \mathtt{P}' \rangle} \langle \mathtt{G}'', \mathtt{P}'' \rangle$, $(\sigma', \langle \mathtt{G}', \mathtt{P}' \rangle) \in \mathcal{R}_{\mathcal{S}}$, and $(\sigma'', \langle \mathtt{G}'', \mathtt{P}'' \rangle) \in \mathcal{R}_{\mathsf{r}}$.*

We write $\sim_{\mathcal{S}}$ for the largest implementation relation.

The notion of implementation correctness coincides with the one given in [9, Definition 21]. We may introduce one-replica implementation relations, and the associated notion of correctness, just by suitably restricting the context LTS, that is, by requiring b and m in items 1 and 2 above to arise from a root extension instead of an extension.

Example 5.12. *We now look at the* GCounter RDT *and we exhibit a one-replica implementation relation for it.*

Restricting the context LTS for $\mathcal{S}_{GC}$ to its one-replica labels, we find the following three types of transitions

T1: $\langle \mathtt{G}, \mathtt{P}\rangle \xrightarrow{\langle \mathtt{rd}, k\rangle} \langle \mathtt{G}^{\langle \mathtt{rd}, k\rangle}, \mathtt{P}'\rangle$, *arising from a rooted $\langle \mathtt{rd}, k\rangle$-extension* $\mathtt{G} \to \mathtt{G}^{\langle \mathtt{rd}, k\rangle}$.

T2: $\langle \mathtt{G}, \mathtt{P}\rangle \xrightarrow{\langle \mathtt{inc}, ok\rangle} \langle \mathtt{G}^{\langle \mathtt{inc}, ok\rangle}, \mathtt{P}'\rangle$, *arising from a rooted $\langle \mathtt{inc}, ok\rangle$-extension* $\mathtt{G} \to \mathtt{G}^{\langle \mathtt{inc}, ok\rangle}$.

T3: $\langle \mathtt{G}, \mathtt{P}\rangle \xrightarrow{\langle \mathtt{G}', \mathtt{P}'\rangle} \langle \mathtt{G}'', \mathtt{P}''\rangle$, *arising from a suitable pushout in* $\mathbf{PDag}(\mathcal{L})$.

Given a RTLS implementation $\mathcal{I}_{\mathcal{S}_{GC}}$, defined according to Example 4.10, let v_{r} denote the state v in $\mathcal{I}_{\mathsf{r}}$. We have that the following family of relations

$$\mathcal{R}_{\mathsf{r}} = \left\{ (v_{\mathsf{r}}, \langle \mathtt{G}, \mathtt{P}\rangle) \;\middle|\; \begin{array}{l} \exists f : \mathcal{E}_{\mathtt{G}} \to \mathcal{R} \text{ s.t.} \\ \forall \mathsf{s} \in \mathcal{R}. v_{\mathsf{r}}(\mathsf{s}) = \sharp\{\mathsf{e} \mid f(\mathsf{e}) = \mathsf{s} \wedge \lambda_{\mathtt{G}} = \langle \mathtt{inc}, ok\rangle\} \end{array} \right\}$$

is a one-replica implementation relation. The proof is a straightforward adaptation of [9, Example 16] and is carried out by case analysis, noting that the transition types T1-T3 above correspond to the tree cases of Definition 5.11.

6. Implementation model

In this section we present our model for implementations. Similarly to what we did for specifications, the aim is to characterise RLTSs by means of suitable functors. Our models are based on a power-domain construction, modelling non-determinism.

6.1. Implementations as functors over power-monoids

Our model for implementations is inspired by [23], where LTSs are given as functors from the free monoid over labels, represented as a one-object category, to a category of non-deterministic computations. This allows modelling sequences of transitions as compositions of computations. In our setting, we use a *one-replica* category representing the free monoid of actions of a replica.

Definition 6.1 (One-replica category)**.** *The* one-replica category **IR** *is the category with one object and whose arrows are words over $\mathcal{M}$.*

Example 6.2. *The one-replica category* **IR** *for the* Multi-value Register *in Ex. 4.14, has a unique object* r *and the arrows* $w : \mathsf{r} \to \mathsf{r}$ *with* $w \in \{\langle \mathtt{wr}(k), ok\rangle \mid k \in \mathcal{V}\}^*$ *with* $\iota_{\mathsf{r}} = \epsilon : \mathsf{r} \to \mathsf{r}$.

In order to capture the behaviour of a set of replicas $\mathcal{R}$, we need to account for the fact that single replicas must show the "same" behaviour. For instance, in the *Multi-value Register* implementation LTS (see Ex. 4.13), the (WRITE) operation on two replicas r and s have to return exactly the same set of version vectors $\mathbb{N}^{\mathcal{R}}$, up to a swapping of r and s in their domain.

We formalise this constraint by introducing the category $\mathbf{IR}(\mathcal{R})$, containing $\#\mathcal{R}$ isomorphic copies of **IR**. That is, that category comes equipped with isomorphisms $\iota_{\mathsf{r},\mathsf{s}} : \mathsf{r} \to \mathsf{s}$ for each $\mathsf{r}, \mathsf{s} \in \mathcal{R}$ such that $\iota_{\mathsf{r},\mathsf{r}} = id_{\mathsf{r}} = \iota_{\mathsf{r},\mathsf{s}}; \iota_{\mathsf{s},\mathsf{r}}$. Furthermore, we require a *naturality* condition, namely, for all words w over $\mathcal{M}$ we have $\iota_{\mathsf{r},\mathsf{s}}; w = w; \iota_{\mathsf{r},\mathsf{s}}$. This constraint precisely enforces the requirement on the behaviour of the single replicas, which are now naturally isomorphic. For the sake of clarity, we usually suffix arrows associated to elements of $\mathcal{M}$ with the replica they belong to, e.g. $m : \mathsf{r} \to \mathsf{r}$ is denoted as m_{r}.

Example 6.3. *Consider again the* Multi-value Register *and two replicas, i.e.,* $\mathcal{R} = \{\mathsf{r}, \mathsf{s}\}$*. Then,* $\mathbf{IR}(\mathcal{R})$ *has two objects* r *and* s *and the arrows* $w : \mathsf{r}_1 \to \mathsf{r}_2$ *with* $\mathsf{r}_1, \mathsf{r}_2 \in \mathcal{R}$ *and* $w \in \{\langle \mathtt{wr}(k), ok\rangle \mid k \in \mathcal{V}\}^*$*. We identify* $\iota_{\mathsf{r}_1,\mathsf{r}_2}$ *with* $\epsilon : \mathsf{r}_1 \to \mathsf{r}_2$ *for* $\mathsf{r}_1, \mathsf{r}_2 \in \mathcal{R}$*. We write* $\langle \mathtt{wr}(k), ok\rangle_{\mathsf{r}}$ *for the arrow* $\langle \mathtt{wr}(k), ok\rangle : \mathsf{r} \to \mathsf{r}$*, i.e., the execution of the mutator* $\langle \mathtt{wr}(k), ok\rangle$ *over the replica* r*. Hence,* $\langle \mathtt{wr}(k_1), ok\rangle_{\mathsf{r}}; \langle \mathtt{wr}(k_2), ok\rangle_{\mathsf{s}}$ *stands for a computation that consists of two writes: Firstly,* k_1 *is written over* r*, then,* k_2 *is written over* s*.*

We now move to define a category where the arrows of $\mathbf{IR}(\mathcal{R})$ are interpreted as non-deterministic computations. The key idea is to generalise the development of Section 4.1 and represent the state-space as a monoid, and transitions as monoid endomorphisms. The first step is to consider the category **Mon** of monoids and monoid homomorphisms. This is not enough, since here the arrows of $\mathbf{IR}(\mathcal{R})$ would be interpreted as functions mapping each state to a single state, i.e., as deterministic computations. To represent nondeterministic computations, we consider the *powerset functor* $\mathbf{P} \colon \mathbf{Set} \to \mathbf{Set}$, mapping a set to its subsets. It is well-known that this functor forms a monad (here we omit unit and multiplication), and that it lifts to monoids, i.e., to a functor $\mathbf{P} \colon \mathbf{Mon} \to \mathbf{Mon}$. We can then consider the associated *Kleisli* category $\mathbf{P}(\mathbf{Mon})$, where objects are those of **Mon** and arrows $\mathcal{M}_1 \to \mathcal{M}_2$ are monoid homorphisms $\mathcal{M}_1 \to \mathbf{P}(\mathcal{M}_2)$, representing relations that respect the monoid structure. In our setting, these will be used to represent transition relations of monoidal RLTSs.

Definition 6.4 (Functorial implementation)**.** *A* functorial implementation *of* $\mathcal{R}$ *is a functor* $\mathbb{I} \colon \mathbf{IR}(\mathcal{R}) \to \mathbf{P}(\mathbf{Mon})$.

An implementation functor thus maps each replica into isomorphic monoids of possible states, and the arrows of a replica are mapped to relations over such sets of states, with the additional property of being compositional with respect to state composition. More precisely, $\mathbb{I}(m_{\mathsf{r}})(\sigma)$ is the set of states of the replica r that are reachable from σ after observing m_{r}. The naturality of the isomorphisms $\iota_{\mathsf{r},\mathsf{s}}$ ensures that the replicas exhibit the "same" behaviour, yet up to isomorphism, which takes care of the possible permutations among replicas.

Example 6.5. *An implementation functor for the* Multi-value Register *over a set of replicas $\mathcal{R}$ can be defined as follows*

- *We assume that all replicas are implemented in the same way (i.e., they only differ in their identifier). For this reason, their states exhibit the same structure, i.e., the implementation monoid $\langle \Sigma, \oplus, 1\rangle$ defined in Example 4.13. Hence, the implementation functor maps each replica is mapped to same implementation monoid, i.e., $\forall \mathsf{r}.\mathbb{I}(\mathsf{r}) = \langle \Sigma, \oplus, 1\rangle$.*

- *Each arrow $\langle \mathtt{wr}(k), ok\rangle : \mathsf{r} \to \mathsf{r}$ corresponding to the execution of the mutator $\langle \mathtt{wr}(k), ok\rangle$ over the replica r is mapped to a transition relation corresponding to that replica, i.e.,*

$$\mathbb{I}(\langle \mathtt{wr}(k), ok\rangle_{\mathsf{r}})(S, v) = \{(S', v') \mid (S, v) \overset{\langle \mathtt{wr}(k), ok\rangle}{\rightsquigarrow}{}^{\mathsf{r}} (S', v'),\\ S' = (\mathsf{r}, v(\mathsf{r}) + 1, k), v' = v[\mathsf{r} \mapsto v(\mathsf{r}) + 1]\}$$

- $\mathbb{I}(\iota_{\mathsf{r},\mathsf{s}}) = \varphi_{\mathsf{r},\mathsf{s}}$ *are RLTS isomorphisms.*

We are now ready to give the main correspondence theorem of this section.

Theorem 6.6. *There is a one-to-one correspondence between monoidal, state-deterministic RLTS implementations and functorial implementations.*

Proof. We first show that a monoidal, state-deterministic RLTS implementation $\mathcal{I}$ over $\mathcal{R}$ induces a functor $\mathbb{I}(\mathcal{I})\colon \mathbf{IR}(\mathcal{R}) \to \mathbf{P}(\mathbf{Mon})$. Suppose $\mathcal{I} = \{\langle \Sigma^{\mathsf{r}}, \oplus^{\mathsf{r}}, 1^{\mathsf{r}}, \rightsquigarrow^{\mathsf{r}}\rangle\}_{\mathsf{r}\in\mathcal{R}}$, and let $\varphi_{\mathsf{r},\mathsf{s}}$ be the isomorphism between $\mathcal{I}_{\mathsf{r}}$ and $\mathcal{I}_{\mathsf{s}}$. Then we define

$$\mathbb{I}(\mathcal{I})(\mathsf{r}) = \langle \Sigma^{\mathsf{r}}, \oplus^{\mathsf{r}}, 1^{\mathsf{r}}\rangle$$
$$\mathbb{I}(\mathcal{I})(m_{\mathsf{r}})(\sigma) = \{\sigma' \mid \sigma \overset{m}{\rightsquigarrow}{}^{\mathsf{r}}_{\mathcal{M}}\, \sigma'\}$$
$$\mathbb{I}(\mathcal{I})(\iota_{\mathsf{r},\mathsf{s}}) = \varphi_{\mathsf{r},\mathsf{s}}$$

with the obvious identity mappings. This functor is well-defined, in fact: $\mathbb{I}(\mathcal{I})(m_{\mathsf{r}})$ is a monoid homomorphism by monoidality of $\mathcal{I}$; $\mathbb{I}(\mathcal{I})(m_{\mathsf{r}}; m'_{\mathsf{r}}) = \mathbb{I}(\mathcal{I})(m_{\mathsf{r}}); \mathbb{I}(\mathcal{I})(m'_{\mathsf{r}})$, as both sides amount to making a m transition followed by a m' transition in $\mathcal{I}_{\mathsf{r}}$; and finally, the equations $\iota_{\mathsf{r},\mathsf{s}}; w = w; \iota_{\mathsf{r},\mathsf{s}}$ in $\mathbf{IR}(\mathcal{R})$ are preserved, because they amount to $\mathbb{I}(\mathcal{I})(\iota_{\mathsf{r},\mathsf{s}}) = \varphi_{\mathsf{r},\mathsf{s}}$ being a RLTS homomorphism, hence commuting with the transition structures of $\mathcal{I}_{\mathsf{r}}$ and $\mathcal{I}_{\mathsf{s}}$.

Although $\mathbb{I}(\mathcal{I})$ does not explicitly represent transitions labelled by states (these do not occur as arrows of $\mathbf{IR}(\mathcal{R})$), they are implicitly represented in the target category $\mathbf{P}(\mathbf{Mon})$ as the monoidal structure. In fact, since the $\mathcal{I}_{\mathsf{r}}$ are all state-deterministic, $\rightsquigarrow$ is the closure of $\rightsquigarrow_{\mathcal{M}}$ with respect to the monoidal operation.

We conclude by observing that the correspondence given above is actually bijective: two distinct RLTSs will yield two distinct functors, and every functor $\mathbf{IR}(\mathcal{R}) \to \mathbf{P}(\mathbf{Mon})$ can be turned into a RLTS using the equations above. $\square$

6.2. From functorial implementations to LTSs

We can distill an LTS from a functorial implementation similarly to how we obtained context LTSs from functorial specifications. Given a functorial implementation $\mathbb{I}$, we can define its category of elements $\mathbf{E}(\mathbb{I})$ as in Definition 5.1. It is rewritten here for the sake of clarity.

Definition 6.7 (Category of elements, II)**.** *The category of elements* $\mathbf{E}(\mathbb{I})$ *of* $\mathbb{I}$ *is obtained as*

- *states are pairs* $\langle \mathsf{r}, \sigma \rangle$*, such that* $\mathsf{r} \in \mathcal{R}$ *and* $\sigma \in \mathbb{I}(\mathsf{r})$*;*
- *arrows* $g : \langle \mathsf{r}, \sigma \rangle \to \langle \mathsf{s}, \sigma' \rangle$ *are arrows* $g : \mathsf{r} \to \mathsf{s}$ *such that* $\sigma' \in \mathbb{I}(g)(\sigma)$*.*

It suffices to apply a simpler machinery than the one used for context LTSs.

Definition 6.8 (Implementation LTS)**.** *The implementation LTS is obtained by taking elements* $\langle \mathsf{r}, \sigma \rangle$ *of* $\mathbf{E}(\mathbb{I})$ *as states, and by labelled transitions triples*

$$\langle \mathsf{r}, \sigma \rangle \xrightarrow{m_{\mathsf{r}}} \langle \mathsf{r}, \sigma' \rangle$$

such that $m_{\mathsf{r}} : \langle \mathsf{r}, \sigma \rangle \to \langle \mathsf{r}, \sigma' \rangle$ *is an arrow of* $\mathbf{E}(\mathbb{I})$*.*

We restricted to transitions over the same replica, but of course this is just for convenience, since all our examples fit into this pattern.

We note that an implementation LTS is in general smaller than the corresponding RLTS, as it lacks explicit state-labelled transitions; these instead are captured by the monoidal structure of the state-space.

7. Conclusions and further works

In our paper we considered RDTs, and we laid out the basis for an algebraic characterisation of their operational semantics as well as of their implementation correctness in terms of (higher-order) simulation. The core of our contribution lies precisely in the formalism behind such characterisations. Our proposal builds on [12, 10] and improves [11, 9] and similar set-theoretical characterisations, which are now made precise and recast into standard notions from the literature on algebraic specifications, thus allowing for the use of a large body of methods and techniques in the analysis of RDTs. We offered a few examples for showing the adequateness of our proposal, even if its strength need to be further checked by a larger number of case studies.

In order to stress the methodological points, we adopted some simplifications. The most notable is the removal of the `snd` label from our transition systems.

Indeed, in our examples, and, in in fact, in most case studies we are aware of, a replica always spawns a full copy of itself, thus from the point of view of simulation it is irrelevant, and it would be in any case captured by the identity arrow on the category of replicas. The modelling of replica communication [9], where the action of sending plays a larger role, is the subject of ongoing work.

Our construction of transition systems out of a category of elements follows an already established pattern for presheaves and simulation, most notably in [23]. The distilling of labels is clearly reminiscent of the *contexts as labels* paradigm advanced by Leifer and Milner [17], and it would fit in its less constrained version proposed in [4]. Since this was not the main methodological issue of the paper, we adopted a presentation requiring some ingenuity.

Our set-theoretical definition of the operational semantics of an implementation focuses on the algebraic structure of replica states. In this respect, it shares aims with the approach presented in [1] for implementation of *Commutative Replicated Data Types* (CRDTs). There, the chosen structure for states are join-semilattices, instead of monoids. As a consequence, mutators in [1] are required to be inflations on the applied states while mutators in our setting must preserve the monoidal structure of the states. As far as we know, the approach presented in this paper is the first one that exploits a monoidal structure for LTSs to obtain a functorial characterisation of implementations.

References

[1] Baquero, C., Almeida, P.S., Cunha, A., Ferreira, C.: Composition in state-based replicated data types. Bullettion of the EATCS **123** (2017)

[2] Bieniusa, A., Zawirski, M., Preguiça, N.M., Shapiro, M., Baquero, C., Balegas, V., Duarte, S.: An optimized conflict-free replicated set. CoRR **abs/1210.3368** (2012)

[3] Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic, Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press (2001)

[4] Bonchi, F., Gadducci, F., Monreale, G.V.: A general theory of barbs, contexts, and labels. ACM Transactions on Computational Logic **15**(4), 35:1–35:27 (2014)

[5] Bouajjani, A., Enea, C., Hamza, J.: Verifying eventual consistency of optimistic replication systems. In: Jagannathan, S., Sewell, P. (eds.) POPL 2014. pp. 285–296. ACM (2014)

[6] Burckhardt, S., Gotsman, A., Yang, H.: Understanding eventual consistency. Tech. Rep. MSR-TR-2013-39, Microsoft Research (2013)

[7] Burckhardt, S., Gotsman, A., Yang, H., Zawirski, M.: Replicated data types: specification, verification, optimality. In: Jagannathan, S., Sewell, P. (eds.) POPL 2014. pp. 271–284. ACM (2014)

[8] Cerone, A., Bernardi, G., Gotsman, A.: A framework for transactional consistency models with atomic visibility. In: Aceto, L., de Frutos-Escrig, D. (eds.) CONCUR 2015. LIPIcs, vol. 42, pp. 58–71. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015)

[9] Gadducci, F., Melgratti, H., Roldán, C.: On the semantics and implementation of replicated data types. Science of Computer Programming **167**, 91–113 (2018)

[10] Gadducci, F., Melgratti, H., Roldán, C., Sammartino, M.: Implementation correctness for replicated data types, categorically. In: Pun, V.K.I., Stolz, V., Simão, A. (eds.) ICTAC 2020. pp. 283–303. Springer (2020)

[11] Gadducci, F., Melgratti, H.C., Roldán, C.: A denotational view of replicated data types. In: Jacquet, J., Massink, M. (eds.) COORDINATION 2017. LNCS, vol. 10319, pp. 138–156. Springer (2017)

[12] Gadducci, F., Melgratti, H.C., Roldán, C., Sammartino, M.: A categorical account of replicated data types. In: Chattopadhyay, A., Gastin, P. (eds.) FSTTCS 2019. LIPIcs, vol. 150, pp. 42:1–42:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)

[13] Gotsman, A., Burckhardt, S.: Consistency models with global operation sequencing and their composition. In: Richa, A.W. (ed.) DISC 2017. LIPIcs, vol. 91, pp. 23:1–23:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)

[14] Gotsman, A., Yang, H.: Composite replicated data types. In: Vitek, J. (ed.) ESOP 2015. LNCS, vol. 9032, pp. 585–609. Springer (2015)

[15] Kaki, G., Earanky, K., Sivaramakrishnan, K.C., Jagannathan, S.: Safe replication through bounded concurrency verification. In: OOPSLA 2018. PACMPL, vol. 2, pp. 164:1–164:27. ACM (2018)

[16] Lanese, I., Pérez, J.A., Sangiorgi, D., Schmitt, A.: On the expressiveness and decidability of higher-order process calculi. Information and Computation **209**(2), 198–226 (2011)

[17] Leifer, J.J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer (2000)

[18] MacLane, S., Moerdijk, I.: Sheaves in geometry and logic: A first introduction to topos theory. Springer (1992)

[19] Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 685–695. Springer (1992)

[20] Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Défago, X., Petit, F., Villain, V. (eds.) SSS 2011. LNCS, vol. 6976, pp. 386–400. Springer (2011)

[21] Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: A comprehensive study of convergent and commutative replicated data types. Tech. Rep. RR-7506, Inria–Centre Paris-Rocquencourt (2011)

[22] Sivaramakrishnan, K.C., Kaki, G., Jagannathan, S.: Declarative programming over eventually consistent data stores. In: Grove, D., Blackburn, S. (eds.) PLDI 2015. pp. 413–424. ACM (2015)

[23] Sobociński, P.: Relational presheaves, change of base and weak simulation. Computer and System Sciences **81**(5), 901–910 (2015)

[24] Zawirski, M., Baquero, C., Bieniusa, A., Preguiça, N.M., Shapiro, M.: Eventually consistent register revisited. In: Alvaro, P., Bessani, A. (eds.) PaPoC@EuroSys 2016. pp. 9:1–9:3. ACM (2016)