Implementation correctness for Replicated Data Types, categorically [⋆]

Fabio Gadducci[1], Hernán Melgratti[2], Christian Roldán[3], and Matteo Sammartino[4]

[1] Dipartimento di Informatica, Università di Pisa
[2] ICC – Universidad de Buenos Aires – CONICET, Argentina
[3] IMDEA Software Institute
[4] 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: Only recently different specification flavours have been proposed, among them a denotational formalism that inter alia captures specification refinement. So far, however, no abstract model has been proposed for the implementations and their correctness with respect to specifications. This paper fills the gap: We first give categorical constructions for distilling an operational model from a specification, as well as its implementations, and then we define a notion of implementation correctness via simulation.

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

1 Introduction

Replicated data types (RDTs) are abstractions for building distributed systems on top of weak-consistent stores, i.e., systems that tolerate temporary data inconsistencies to favour availability. Different specification approaches for RDTs have been proposed in the literature [2–5, 7, 8, 10, 11, 14, 16]. 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. Consider an RDT *Counter*, which has 0 as initial value and the operations `inc` to increment it and `rd` to read its value. By following the functional approach proposed in [8, 7], the RDT *Counter* is specified as a function $\mathcal{S}_{Ctr}$ that

[⋆] 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, by the PIP project 11220130100148CO and by the Leverhulme Prize PLP-2016-129.

$$\mathcal{S}_{Ctr}\left(\begin{tikzcd}[column sep=small] \langle \texttt{inc}, ok\rangle \arrow[d] & \langle \texttt{inc}, ok\rangle \arrow[dl] \arrow[d] \\ \langle \texttt{rd}, 2\rangle & \langle \texttt{rd}, 1\rangle \end{tikzcd}\right) = \left\{\begin{tikzcd}[column sep=small, row sep=small] \langle \texttt{inc}, ok\rangle \arrow[d, dash] & \langle \texttt{rd}, 1\rangle \arrow[d, dash] & \ldots \\ \langle \texttt{rd}, 1\rangle \arrow[d, dash] & \langle \texttt{inc}, ok\rangle \arrow[d, dash] & \\ \langle \texttt{inc}, ok\rangle \arrow[d, dash] & \langle \texttt{inc}, ok\rangle \arrow[d, dash] & \\ \langle \texttt{rd}, 2\rangle & \langle \texttt{rd}, 2\rangle & \end{tikzcd}\right\} \quad \mathcal{S}_{Ctr}\left(\begin{tikzcd} \langle \texttt{inc}, ok\rangle \arrow[d] \\ \langle \texttt{rd}, 0\rangle \end{tikzcd}\right) = \emptyset$$

Fig. 1. *Counter* specification.

maps visibilities into sets of arbitrations, both of them represented as directed graphs labelled by pairs $\langle \textit{operation}, \textit{result}\rangle$. Figure 1 illustrates two cases for the definition of $\mathcal{S}_{Ctr}$. The left-most equation considers the case in which the state consists of four operations: two `inc`s that return *ok* and two `rd`s that respectively returns 1 and 2. The arrows in the visibility (i.e., the graph on the left-hand side of the equation) denotes the fact that the execution of one `rd` sees the effects of the execution of the two `inc`s while the other sees just one of them. Intuitively, the arrows justify the values returned by each `rd`. In this case, $\mathcal{S}_{Ctr}$ maps that visibility graph into a non-empty set of arbitrations, i.e., a set of total orders of executed operations. While the leftmost arbitration is immediate, the rightmost one shows that arbitrations do not necessarily preserve the order of the visibility graph: While this may seem counterintuitive, it is a design choice that allows more flexibility and is adopted by most of the current proposals for RDTs. The situation illustrated by the equation on the right is different, because the result of `rd` is defined as 0, which is deemed inconsistent with the fact that the execution of that `rd` sees one `inc`. For this reason, $\mathcal{S}_{Ctr}$ maps such visibility into an empty set of arbitrations (i.e., the visibility describes an unreachable state).

As shown in [9], a large class of functional specifications, dubbed *coherent*, can be characterised as functors between the categories $\mathbf{PIDag}(\mathcal{L})$ of visibilities and $\mathbf{SPath}(\mathcal{L})$ of sets of arbitrations, where $\mathcal{L}$ is a fixed set of operations labels. In this paper, we take advantage of the functorial characterisation of specifications to develop a notion of implementation and implementation correctness for RDTs. We first provide a systematic way for recovering an operational semantics out of a specification. This is achieved by constructing the *category of elements* $\mathbf{E}(\mathbb{F})$ of the functor $\mathbb{F}$ associated with a specification. Objects in $\mathbf{E}(\mathbb{F})$ are pairs $\langle \mathtt{G}, \mathtt{P}\rangle$ describing the states of the RDT in terms of a visibility $\mathtt{G}$ and an arbitration $\mathtt{P}$. Arrows of $\mathbf{E}(\mathbb{F})$ stand for computations. Then, by following the approach in [12], we recover an LTS from $\mathbf{E}(\mathbb{F})$ by assigning labels to the computations (arrows) of $\mathbf{E}(\mathbb{F})$: The labels provide the contextual information that explains the way in which a local computation is embedded into a global context. The obtained semantics is then an operational specification for the implementation of the RDT. Usually, RDTs are implemented by several replicas that keep their own local state and propagate changes asynchronously. Such behaviour can be understood in terms of two labelled transition systems (LTSs): One that describes the behaviour of a single replica, and another one, obtained by composition, that accounts for the concurrent execution of several interacting replicas. Our first observation is that usual implementations of well-known RDTs exhibit a preordered monoidal structure on states, where the order accounts for the evolution of the system and the monoidal operator for state composition. Consequently, the computation space of a replica can be defined in terms of the power-domain construction over the corresponding monoid. Then, the behaviour of a replica can be represented by a functor that maps sequences of operations into computations. The category of elements associated with the implementation functor is used again to recover a contextual LTS, in this case, for an implementation. Consequently, implementation correctness can be straightforwardly stated in terms of contextual simulation between the recovered LTSs. In this way, we reframe previous ad-hoc formulations of implementation correctness by applying well-known notions in concurrency theory, thus paving the way for the application of more standard techniques in the analysis of RDTs.

The paper is structured as follows. Sect. 2 recalls the functorial presentation of RDTs specifications [9]. Sect. 3 presents the basics of state-based implementation of RDTs and the running examples. Sect. 4 recasts the set-theoretical presentation [7] of the operational semantics of RDTs in terms of categories of elements and introduces contextual LTSs. Sect. 5 illustrates a categorical model for implementations and characterises their correctness via simulation relations.

2 Background

Notation. Given a finite set $\mathtt{E}$, a (binary) *relation* ρ over $\mathtt{E}$, written $\langle \mathtt{E}, \rho \rangle$, is a subset $\rho \subseteq \mathtt{E} \times \mathtt{E}$. We write $\emptyset$ for the empty relation and $\mathtt{e}\ \rho\ \mathtt{e}'$ to mean $(\mathtt{e}, \mathtt{e}') \in \rho$. A subset $\mathtt{E}' \subseteq \mathtt{E}$ is *downward closed* with respect to ρ if $\mathtt{e}\ \rho\ \mathtt{e}'$ implies $\mathtt{e} \in \mathtt{E}'$, for all $\mathtt{e}' \in \mathtt{E}'$. We write $\lfloor \mathtt{e} \rfloor_\rho$ for the smallest downward closed set with respect to ρ including $e \in \mathtt{E}$, omitting the subscript ρ if clear from the context.

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 $\mathtt{G}$. 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 $\mathtt{P}$).

We denote with $\mathbb{G}(\mathcal{L})$ and $\mathbb{P}(\mathcal{L})$ the collections of (finite) graphs and (finite) paths, respectively, labelled on $\mathcal{L}$ and with ϵ the empty graph. 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 vertices are those in $\mathcal{E}$ and are labelled by $\lambda : \mathcal{E} \to \mathcal{L}$.

2.1 Replicated data types

We briefly recall the functional model of RDTs introduced in [7].

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

Fig. 2. Specification of a multi-value Register

Definition 1 (Specifications). *A* specification $\mathcal{S}$ *is a 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 $\mathtt{G}$.

Example 1 (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

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

where $\langle \texttt{wr}(k), ok\rangle$ stands for an operation that writes the integer k and $\langle \texttt{rd}, S\rangle$ 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

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

The condition on the right requires that any event $\mathtt{e}$ in $\mathtt{G}$ associated with a read (i.e., labelled by $\langle \texttt{rd}, S\rangle$) 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 $\mathtt{P}$, otherwise $\mathcal{S}_{mvR}(\mathtt{G}) = \emptyset$.

An instance of $\mathcal{S}_{mvR}$ is shown in Fig. 2, where $\mathtt{G}$ consists of three writes: $\texttt{wr}(2)$ overwrites $\texttt{wr}(1)$, and both are concurrent with $\texttt{wr}(3)$. Additionally, there are two reads: One observes all writes (right-most at the bottom), the other does not see $\texttt{wr}(3)$ (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. 2 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. 2). This a common approach in the specification of RDTs [4] because it allows permissive strategies for implementation (we refer to [7] for details). If needed, a specification can explicitly exclude arbitrations (as illustrated in Ex. 2).

Example 2 (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

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

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

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

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

We now restrict our attention to *coherent* specifications, which suffice for the standard specification of RDTs [7] and are amenable to 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 (Product). *The product of a set $\mathcal{X}$ of compatible paths is*

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

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. A set of sets of paths $\mathcal{X}_1, \mathcal{X}_2, \ldots$ is compatible if $\bigcup_i \mathcal{X}_i$ is so, and we can define $\bigotimes_i \mathcal{X}_i$ as $\bigotimes \bigcup_i \mathcal{X}_i$.

Definition 3 ((Past-)Coherent Specification). *Let $\mathcal{S}$ be a specification. We say that $\mathcal{S}$ is past-coherent (briefly, coherent) if $\forall \mathtt{G} \neq \epsilon.\ \mathcal{S}(\mathtt{G}) = \bigotimes_{\mathtt{e}\in\mathcal{E}_{\mathtt{G}}} \mathcal{S}(\mathtt{G}|_{\lfloor \mathtt{e}\rfloor})$.*

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 $\mathtt{G}|_{\lfloor \mathtt{e}\rfloor}$. It can be shown that the specifications in Ex. 1 and 2 are coherent.

2.2 Categorical model of specifications

We now recall important definitions and results from [9]. We start off by introducing the category of binary relations.

Definition 4 ((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. The intuition is that morphisms add no dependencies in the past of an event, hence the chosen name.

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 both finite limits and finite colimits, which are computed point-wise as in **Set**. The structure is largely lifted to **PBin**: Finite colimits and binary pullbacks in **PBin** are computed as in **Bin**. Yet there is no terminal object, as morphisms into the singleton are clearly not past-reflecting.

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

Definition 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 inclusion into* **Set**. *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 also past-reflecting. In both categories, finite colimits and binary pullbacks always exist and are essentially computed as in **Bin**. Two sub-categories are used for both the syntax and the semantics of specifications.

Definition 6 (PDag/Path). **PDag** *is the full sub-category of* **PBin** *whose objects are acyclic graphs, and the same for* **Path** *with respect to* **Bin** *and paths.*

As for relations, suitable comma categories capture labelled paths and graphs, which are respectively called $\mathbf{PDag}(\mathcal{L})$ and $\mathbf{Path}(\mathcal{L})$. Once more, finite colimits and binary pullbacks always exist and are essentially computed as in **Bin**.

Example 3. We illustrate some labelled graphs in Fig. 3 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 $\langle \mathtt{wr}(1), \mathit{ok} \rangle$ 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).

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

(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) $\mathtt{G}_2$

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

(d) $\mathtt{G}_3$

Fig. 3. Some labelled graphs

Model of specifications. Specifications are modelled as functors from graphs to sets of paths. *Saturation* is used to define morphisms between sets of paths.

Definition 7 (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* $\bigcup_{\mathtt{P} \in \mathcal{X}} \mathtt{sat}(\mathtt{P}, \mathtt{f})$*.*

Definition 8 (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* $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathcal{X}_1, \mathtt{f})$*.*

There is a ps-morphism from set of paths $\mathcal{X}_1$ to set of paths $\mathcal{X}_2$ if any path in $\mathcal{X}_2$ can be obtained by adding events to a path in $\mathcal{X}_1$. This notion captures the idea that arbitrations of larger visibilities are extensions of smaller visibilities.

Example 4. Consider the label-preserving function $\mathtt{f}_2 : (\mathcal{E}_{\mathtt{P}_1}, \lambda_{\mathtt{P}_1}) \to (\mathcal{E}_{\mathtt{G}_2}, \lambda_{\mathtt{G}_2})$ in Ex. 3. Then $\mathtt{sat}(\mathtt{P}_1, \mathtt{f}_2)$ contains four paths, all of them an extension of $\mathtt{P}_1$ obtained by inserting a new event labelled by $\langle \mathtt{wr}(3), \mathtt{ok}\rangle$ at any arbitrary position. Moreover, there exists a ps-morphism $\mathtt{f} : \{\mathtt{P}_1\} \to \mathcal{X}$ for any $\mathcal{X} \subseteq \mathtt{sat}(\mathtt{P}_1, \mathtt{f}_2)$.

Definition 9 (Sets of Paths Category). $\mathbf{SPath}(\mathcal{L})$ *is the category whose objects are sets of paths labelled over* $\mathcal{L}$ *and arrows are ps-morphisms.*

$\mathbf{PIDag}(\mathcal{L})$ is the sub-category of acyclic graphs and monic pr-morphisms: It lacks pushouts, but these can be computed in $\mathbf{PDag}(\mathcal{L})$. We say that a functor $\mathbb{F}$ *weakly* preserves colimits if any 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 $\mathbf{SPath}(\mathcal{L})$.

We now summarise the completeness results presented in [9], albeit slightly rephrased for the sake of simplicity and technical convenience in our developments. We characterise pr-morphisms that add a single event $\top$ to a graph.

Definition 10 (Extension). *Let* $\mathtt{f} : \langle \mathcal{E}_1, \prec_1, \lambda_1\rangle \to \langle \mathcal{E}_2, \prec_2, \lambda_2\rangle$ *be a mono pr-morphism. It is an* extension *along* ℓ *(shortly,* ℓ-extension*) if* $\mathcal{E}_2 = \mathcal{E}_1 + \{\top\}$*,* $\mathtt{f} : \mathcal{E}_1 \to \mathcal{E}_1 + \{\top\}$ *is the associated injection, and* $\lambda_2(\top) = \ell$*.*

A graph $\mathtt{G}$ is *rooted* if there is an event $\mathtt{e} \in \mathcal{E}_{\mathtt{G}}$ such that $\mathtt{G} = \mathtt{G}|_{\lfloor \mathtt{e} \rfloor}$. An extension $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$ is a root extension if $\mathtt{G}_2$ is rooted. It is analogously defined when a mono ps-morphism $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ is a *ps-extension* (along ℓ).

Example 5. Assuming $\mathtt{f}_2$ and $\mathtt{f}_3$ in Ex. 3 are mono, then 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 $\mathtt{P}_1$. Moreover, the ps-morphisms in Ex. 4 induced by $\mathtt{f}_2$ are ps-extensions (along $\langle \mathtt{wr}(3), ok \rangle$).

Definition 11. *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.*

Coherent functors thus preserve monos. Combined with the simpler definition of extension, now easily applied to paths, it allows for a more general presentation of a key result in previous work [9, Theorem 33] (and an immediate instantiation to Propositions 34 and 35 therein), as stated below.

Theorem 1. *Coherent functors induce coherent specifications, and vice versa.*

The core of the result above can be immediately derived from the lemma below, which will have an interest of its own in the following pages. Intuitively, it 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 3).

Lemma 1. *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* $\mathbf{SPath}(\mathcal{L})$

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

3 State-based 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. 4 illustrates a scenario involving two replicas, namely r_1 and r_2, that implement a *multi-value Register* (as specified in Ex. 1). 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. 2. The two writes on r_1 are totally ordered (events **1** and **3**); consequently, 2 overwrites 1. The remaining write takes place on r_2 (event **2**) and is unknown to r_1 until r_2 propagates its changes. Hence, the first read on r_1 (event **5**) returns 2, which is the last written value in r_1. Replicas communicate their local changes by using primitives $\mathtt{snd}$ and $\mathtt{rcv}$.

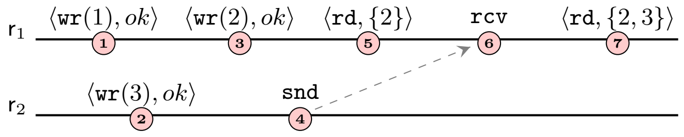
r_1
$\langle \texttt{wr}(1), ok\rangle$
$\langle \texttt{wr}(2), ok\rangle$
$\langle \texttt{rd}, \{2\}\rangle$
rcv
$\langle \texttt{rd}, \{2,3\}\rangle$
1
3
5
6
7
r_2
$\langle \texttt{wr}(3), ok\rangle$
snd
2
4

Fig. 4. Execution

$$\text{(READ)}\ \langle \mathsf{r}, S\rangle \xrightarrow{\texttt{rd},\pi_1 S} \langle \mathsf{r}, S\rangle \qquad \text{(WRITE)}\ \langle \mathsf{r}, S\rangle \xrightarrow{\texttt{wr}(\mathsf{a}),ok} \langle \mathsf{r}, \{(\mathsf{a}, \pi_2 S \triangleright \mathsf{r})\}\rangle$$

$$\text{(SEND)}\ \langle \mathsf{r}, S\rangle \xrightarrow{\texttt{snd},\langle \mathsf{r},S\rangle} \langle \mathsf{r}, S\rangle \qquad \text{(RCV)}\ \langle \mathsf{r}, S\rangle \xrightarrow{\texttt{rcv},\langle \mathsf{r}',S'\rangle} \langle \mathsf{r}, S \oplus S'\rangle$$

Fig. 5. Implementation of data type *multi-value Register*

Event **6** in r_1 denotes the synchronisation of its local state with the state of r_2, i.e., r_1 becomes aware of the written value 3 (depicted by the dashed line between events **4** and **6**). Since writes in r_1 and r_2 are concurrent, the last read on 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 [4] while they only communicate operations (or their effects) under the *operation-based* approach [14]. Hereafter, we will focus on state-based implementations. We write Σ for the set of possible states $\sigma, \sigma_0, \ldots$ of a replica, and define the behaviour of a replica implementing a specification $\mathcal{S} : \mathbb{G}(\mathcal{L}) \to 2^{\mathbb{P}(\mathcal{L})}$ with a labelled transition system $(\Sigma, \mathcal{A}_{\mathcal{S}}, \to)$, where

$$\mathcal{A}_{\mathcal{S}} = \mathcal{L} \cup (\{\texttt{rcv},\ \texttt{snd}\} \times \Sigma)$$

is the set of labels that, in addition to the RDT operations, includes $\langle \texttt{snd}, \sigma\rangle$ and $\langle \texttt{rcv}, \sigma\rangle$ for replica synchronisation.

Example 6. We present an implementation for a *multi-value Register* based on version vectors in [6, 15], in which each replica maintains a set with all maximal concurrent written values. The implementation associates a *version vector* to each written value to determine if two writes are concurrent or causally ordered. A version vector is just a mapping from replicas to natural numbers. Given a set of replicas $\mathcal{R}$, $(\mathbb{N}^{\mathcal{R}}, \leq)$ is the poset of version vectors, where $\leq$ is the standard partial order of a function space, i.e., $\forall v, v' \in \mathbb{N}^{\mathcal{R}}.\ v \leq v$ iff $\forall \mathsf{r} \in \mathcal{R}.\ v(\mathsf{r}) \leq v(\mathsf{r})$. We equip version vectors with an operation $\triangleright : 2^{\mathbb{N}^{\mathcal{R}}} \times \mathcal{R} \to \mathbb{N}^{\mathcal{R}}$ that takes a set of version vectors V and a replica r and generates a new version vector that dominates all elements in V, defined as

$$(V \triangleright \mathsf{r})(\mathsf{r}') = \begin{cases} 1 + \max_0\{v(\mathsf{r}') \mid v \in V\} & \textit{if}\ \mathsf{r}' = \mathsf{r} \\ \max_0\{v(\mathsf{r}') \mid v \in V\} & \textit{if}\ \mathsf{r}' \neq \mathsf{r} \end{cases}$$

where $\max_0$ denotes the maximum of a set with the provision that $\max_0\ \emptyset = 0$.

$$\text{(READ)}\ \langle \mathtt{r}, (t, \mathtt{a})\rangle \xrightarrow{\mathtt{rd},\mathtt{a}} \langle \mathtt{r}, (t, \mathtt{a})\rangle \qquad \text{(WRITE)}\ \langle \mathtt{r}, (t, \mathtt{a})\rangle \xrightarrow{\mathtt{wr(b)},\mathtt{ok}} \langle \mathtt{r}, \max\{(t, \mathtt{a}), (t', \mathtt{b})\}\rangle$$
$$\text{(SEND)}\ \langle \mathtt{r}, (t, \mathtt{a})\rangle \xrightarrow{\mathtt{snd},\langle \mathtt{r},(t,\mathtt{a})\rangle} \langle \mathtt{r}, (t, \mathtt{a})\rangle \quad \text{(RCV)}\ \langle \mathtt{r}, (t, \mathtt{a})\rangle \xrightarrow{\mathtt{rcv},\langle \mathtt{r}',(t',\mathtt{b})\rangle} \langle \mathtt{r}, \max\{(t, \mathtt{a}), (t', \mathtt{b})\}\rangle$$

Fig. 6. Implementation of data type *last-write wins Register*.

Each replica maintains a set $S \in 2^{\mathbb{N}\times\mathbb{N}^{\mathcal{R}}}$ of pairs (n, v), where n is a written value and v is a version vector. We write π_i for the i-th projection of a product (and also for its obvious extension to sets of tuples), and consider $\mathbb{N} \times \mathbb{N}^{\mathcal{R}}$ ordered by $\leq$, where $e \leq e'$ iff $\pi_2 e \leq \pi_2 e'$. Consequently, the maximal concurrent written values in a set $S \in 2^{\mathbb{N}\times\mathbb{N}^{\mathcal{R}}}$ are the *maximal elements* $\|S\|$ defined as $\{u \in S \mid \nexists v \in S, u \leq v\}$. The combination $S_1 \oplus S_2$ of two sets S_1 and S_2 is $\|S_1 \cup S_2\|$.

The behaviour of a replica implementing a *multi-value Register* is defined by the LTS $\langle \Sigma, \mathcal{A}_{\mathcal{S}_{mvR}}, \rightarrow\rangle$ where

- $\Sigma = \mathcal{R} \times \{\|S\| \mid S \in 2^{\mathbb{N}\times\mathbb{N}^{\mathcal{R}}}\}$, i.e., each state consists of the identifier of the replica and a set of maximal concurrent written values;
- $\mathcal{A}_{\mathcal{S}_{mvR}} = \mathcal{L}_{\mathcal{S}_{mvR}} \cup (\{\mathtt{snd},\ \mathtt{rcv}\} \times \Sigma)$, i.e., the set of labels accounts for the operations of the RDT;
- $\rightarrow$ is given by the inference rules in Fig. 5. A read returns the set of locally-stored values (π_1 discards all version vectors), while a write updates the local state with a singleton containing the written value and a version vector that dominates all known values. Rule (SEND) propagates the local state while (RCV) combines the local state with the one received from another replica.

Example 7. We now describe the implementation based on timestamps proposed in [15] for the *last-write wins Register* in Ex. 2. Let $(\mathbb{T}, <)$ be the totally-ordered set of timestamps. Then, the implementation is the LTS $\langle \Sigma, \mathcal{A}_{\mathcal{S}_{lwwR}}, \rightarrow\rangle$ where

- $\Sigma = \mathcal{R} \times (\mathbb{T} \times (\mathbb{N} \cup \{\bot\}))$, i.e., the state $\langle \mathtt{r}, (t, \mathtt{a})\rangle$ of a replica $\mathtt{r}$ contains the current value $\mathtt{a}$ of the register and its associated timestamp. We establish that $\bot < n$ for all $n \in \mathbb{N}$ and consider $\mathbb{T} \times (\mathbb{N} \cup \{\bot\})$ lexicographically ordered, i.e., $(t, \mathtt{a}) \leq (t', \mathtt{a}')$ when either $t < t'$ or $t = t'$ and $\mathtt{a} \leq \mathtt{a}'$.
- $\mathcal{A}_{\mathcal{S}_{lwwR}} = \mathcal{L}_{\mathcal{S}_{lwwR}} \cup (\{\mathtt{snd},\ \mathtt{rcv}\} \times \Sigma)$ is the set of the data type operations;
- $\rightarrow$ is given by the inference rules in Fig. 6. As for the *multi-value Register*, a read just retrieves the value stored in the replica but does not alter its state. A write may change the state of the replica by picking the maximum pair according to lexicographic order. The timestamp t' on the right-hand-side of the rule (WRITE) should be understood as any $t' \in \mathbb{T}$. Rules (SEND) and (RCV) are analogous to the previous example.

4 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 construct its *category of elements*, which is reminiscent of the category of elements for a presheaf (see e.g. [13, Chapter 5]). Following this analogy, given a pr-morphism $\mathtt{f} \colon \mathtt{G} \to \mathtt{G}_1$ and $\mathtt{P} \in \mathbb{F}(\mathtt{G})$, we denote $\mathbb{F}(\mathtt{f})(\mathtt{P})$ 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 12 (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 [17, Definition 4.1].

Example 8. Consider the functor $\mathbb{M}(\mathcal{S}_{mvR})$ induced by the coherent specification $\mathcal{S}_{mvR}$ in Ex. 1. 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. 3c and Fig. 3d, 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. 3c 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 Ex. 2. 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. 3a 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 $\mathcal{S}_{mvR}$.

4.1 One- and Multi-replica LTSs

In [7, Definition 16] an LTS modelling the operational behaviour of a single replica – *one-replica* in short – is derived from a specification as follows

$$\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\ell} \langle \mathtt{G}_1, \mathtt{P}_1 \rangle \quad \iff \quad \mathtt{G}_1 = \mathtt{G}^{\ell},\ \mathtt{P}_1|_{\mathcal{E}_{\mathtt{G}}} = \mathtt{P}$$

where $\mathtt{G}^{\ell}$ is the root extension along ℓ of $\mathtt{G}$. That is, a pair evolves to one where the visibility relation is augmented with a top event labelled ℓ and the path is obtained by adding the new event to $\mathtt{P}$. This way of augmenting the visibility 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.

Lemma 2. *Let $\mathbf{E}_{\circ}(\mathbb{F})$ 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 $\mathbf{E}_{\circ}(\mathbb{F})$.*

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.

The next step is to characterise *multi-replica* LTS as in [7, Definition 20], which model multiple replica evolving concurrently. We recall the main concepts. Suppose we have two replica, and the current state for each is $\langle \mathtt{G}_i, \mathtt{P}_i \rangle$. Let us further assume that $\mathtt{G}_1$ and $\mathtt{G}_2$ are compatible [7, Definition 19], i.e., there is 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). Finally, let $\mathtt{G}_1 \sqcup \mathtt{G}_2$ (which is just set-theoretical union) and the obvious morphisms be the pushout. Then 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. The multi-replica LTS is derived from the one-replica LTS by adding the following inference rule

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

Intuitively, global computations are derived from computations of single replica.

We can recover the multi-replica LTS by exploiting the structure of coherent functors. We need two technical lemmata. 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 3 (Decomposition). *Consider the following diagrams in* $\mathbf{SPath}(\mathcal{L})$

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

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 4. *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 precisely 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 1. *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* $\mathbf{E}_{\mathtt{m}}(\mathbb{F})$*.*

Example 9. Consider once more the category $\mathbb{M}(\mathcal{S}_{mvR})$ discussed in Ex. 8. By Lem. 2, 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 Ex. 8 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 Lem. 4: 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.

4.2 Contextual 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 [12], 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 13. *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 constructively obtained, 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. *The LTS for $\mathbf{E}_{\mathtt{o}}(\mathbb{F})$ ($\mathbf{E}_{\mathtt{m}}(\mathbb{F})$) coincides with the restriction of the contextual 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 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 Ex. 9. 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 Implementation model

In this section we present our model for implementations. Similarly to what we have done for specifications, the aim is to obtain implementation LTSs as the category of elements of suitable functors. Our models are based on a power-domain construction, modelling non-determinism. We will show that we can capture several RDTs, and characterise implementation correctness via simulation.

5.1 Implementations as functors over power-domains

Our model for implementations is inspired by [17], where LTSs are modelled as functors from the free monoid over labels, represented as a one-object category, to a suitable category of non-deterministic computations. This allows modelling sequences of transitions as compositions of computations. In our setting, we introduce a *one-replica* category representing the free monoid of labels of a replica.

Definition 14 (One-replica category). *The* one-replica category $\mathbf{IR}$ *is the category with one object, and where morphisms are words over* $\mathcal{L} \cup \{\mathtt{rcv}\}$.

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. 6), 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 $\mathbf{IR}$. That is, that category comes equipped with isomorphisms $\iota_{r,s} : r \to s$ for each $r, s \in \mathcal{R}$ such that $\iota_{r,r} = id_r = \iota_{r,s}; \iota_{s,r}$. Furthermore, we require a *naturality* condition, namely, for all words w over $\mathcal{L} \cup \{\mathtt{rcv}\}$ we have $\iota_{r,s}; w = w; \iota_{r,s}$. This constraint precisely enforces the requirement on the behaviour of the single replicas, which are now the same up to naturality. For the sake of clarity, we usually suffix arrows associated to elements of $\mathcal{L} \cup \{\mathtt{rcv}\}$ with the replica they belong to, e.g. $\ell : r \to r$ is denoted as ℓ_r.

We now move to define a category where the arrows of $\mathbf{IR}(\mathcal{R})$ are interpreted as non-deterministic computations. Here we assume a category $\mathbf{M}$ where objects are states and arrows stand for sets of *deterministic* computations. We shall see later how to instantiate $\mathbf{M}$ for RDTs.

Definition 15 (Power-domain category). *Let* $\mathbf{M}$ *be a small category. Then, its* power-domain $\mathbf{P}(\mathbf{M})$ *is the category whose objects are sets of objects of* $\mathbf{M}$ *and arrows are pairs* $(R, \{f_i\}_{i \in R}) : X \to Y$ *such that* $R \subseteq X \times Y$ *is a relation and* $\{f_i\}_{i \in R}$ *is a family (indexed by pairs* $\langle x, y \rangle$ *in* R*) of non-empty sets of arrows in* $\mathbf{M}$ *such that* $f_{\langle x,y \rangle} \subseteq Hom_{\mathbf{M}}[x, y]$. *If* $\mathbf{M}$ *is (symmetric) monoidal, so is* $\mathbf{P}(\mathbf{M})$.

An element of $f_{\langle x,y \rangle}$ is thus an arrow in $\mathbf{M}$ from x to y. For simplicity, we often denote $\{f_i\}_{i \in R}$ as f_R. Also, given element $x \in X$ and morphism $(R, f_R) : X \to Y$, we denote as $(R, f_R)(x)$ the set $\{y \mid \langle x, y \rangle \in R\}$.

Definition 16 (Implementation). *Let* $\mathbf{M}$ *be a category. An* implementation *of* $\mathcal{R}$ *in* $\mathbf{M}$ *is a functor* $\mathbb{I}\colon \mathbf{IR}(\mathcal{R}) \to \mathbf{P}(\mathbf{M})$ *such that* $\mathbb{I}(r) = \mathbb{I}(s)$ *for all* $r, s \in \mathcal{R}$.

An implementation functor thus maps each replica into the same set of possible states S, and the arrows of a replica are (morally) mapped to relations over $S \times S$. More precisely, $\mathbb{I}(g)(x)$ is the set of states that are reachable from x after observing g; this will be used later on to synthesise the corresponding LTS. The naturality of the isomorphisms $\iota_{r,s}$ guarantees that the replicas exhibit the "same" behaviour, yet only up to isomorphism, which takes care of the possible permutations among replicas.

5.2 From implementations to replica LTSs

We now define the category of elements $\mathbf{E}(\mathbb{I})$ for an implementation $\mathbb{I}$ as in Definition 12. It is rewritten here for the sake of clarity.

Definition 17 (Category of elements, II). *The category of elements* $\mathbf{E}(\mathbb{I})$ *of* $\mathbb{I}$ *is obtained as*

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

It now suffices to apply the machinery used for obtaining context LTSs.

Definition 18 (Implementation LTS). *The implementation LTS is obtained by taking elements* $\langle r, m\rangle$ *of* $\mathbf{E}(\mathbb{I})$ *as states, and by labelled transitions triples*

$$\langle r, m\rangle \xrightarrow{\langle g, \overline{g}\rangle} \langle r, n\rangle$$

such that $g : \langle r, m\rangle \to \langle r, n\rangle$ *and* $\overline{g} : \langle s, o\rangle \to \langle r, n\rangle$ *are arrows 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. Also, note that each arrow $g : \langle r, m\rangle \to \langle r, n\rangle$ of $\mathbf{E}(\mathbb{I})$ induces at least one labelled transition.

5.3 Deterministic computations

Given a state, what we need is the possibility for it to a) evolve towards a different state and b) be combined with other states. We formalise these ideas via *pre-ordered monoids*, i.e., structures $\mathcal{M} = \langle M, \leq, \otimes, 1\rangle$ consisting of a set M, a pre-order $\leq \subseteq M \times M$, and an associative binary operation $\otimes \colon M \times M \to M$ with an identity element $1 \in M$ such that $1 \leq x$ and $x \leq y$ implies $x \otimes w \leq y \otimes w$.[5]

Remark 1. In order to take into account the isomorphic behaviour of distinct replicas, the pre-ordered monoid $\mathcal{M}$ should come equipped with a group action that preserves both the order and the monoidal structure of $\mathcal{M}$. However, all our cases will fit the bill. First of all note that given a set N, the cartesian product $\mathcal{M} \times \mathcal{M}$ and the function space $\mathcal{M}^N$ are pre-ordered monoids, with $\mathcal{M}$ a sub-monoid of $\mathcal{M}^N$ and $\mathcal{M}^N \times \mathcal{M}^N$ equal to $(\mathcal{M} \times \mathcal{M})^N$. Also, $\mathcal{M}^N$ is equipped with a group action $\Pi(N) \times \mathcal{M}^N \to \mathcal{M}^N$ given by the permutation group $\Pi(N)$ on N, and if X, Y are sub-monoids of $\mathcal{M}^N$ that are closed under the group action, then also the sub-monoid $X \times Y$ of $(\mathcal{M} \times \mathcal{M})^N$ is closed. Thus, all of our examples fit into the shape $\mathcal{M}^{\mathcal{R}}$, for $\mathcal{R}$ the set of replicas at hand.

[5] Note that this is more general than the lattice of states proposed in [14]. First of all, we consider a pre-order instead of a partial order, and furthermore we do not require $\otimes$ to be induced by $\leq$. This weakening results in an algebraic structure that allows for modelling a large family of RDTs.

Example 11. Let us consider the *multi-value Register* from Ex. 1. Recall that states of a replica r consist of a sets of pairs $(n, v) \in \mathbb{N} \times \mathbb{N}^{\mathcal{R}}$, where n is a written value and v is a version vector. In the implementation (Fig. 5) these sets can evolve to: a) themselves (rules (Read) and (Send); b) to a set containing a pair dominating the source state (rule (Write)) and to a set obtained as the combination of the source state and the received one (rule (Rcv)).

These evolutions can be modeled as a pre-ordered monoid structure over $2^{\mathbb{N}\times\mathbb{N}^{\mathcal{R}}}$, namely $\mathcal{M} = \langle 2^{\mathbb{N}\times\mathbb{N}^{\mathcal{R}}}, \leq, \oplus, \emptyset\rangle$, where $\leq$ and $\oplus$ are defined as in Ex. 6.

Example 12. Consider the implementation of *last-write wins Register* in Ex. 7. The current state of replica r consists of a timestamp $t \in \mathbb{T}$ and the value in the register $\mathsf{a} \in \mathbb{N} \cup \{\bot\}$. The evolution of pairs $\langle t, \mathsf{a}\rangle$ in the implementation is captured via the pre-ordered monoid $\langle \mathbb{T} \times \mathbb{N}_\bot, \leq, \max, \langle\bot, \bot\rangle\rangle$, where $\leq$ stands for lexicographic order and $\max\{p, q\} = q$ if and only if $p \leq q$. This max operation is precisely what is used to define the rules (Write) and (Rcv) in Fig. 6.

Given a pre-ordered monoid $\mathcal{M}$ modelling the deterministic behaviour of replicas, we can easily derive a category of deterministic computations. This is intended to be used as base category for the power domain construction of § 5.1.

Definition 19 (Category of deterministic computations). *The category $\mathcal{C}(\mathcal{M})$ has objects the elements of $\mathcal{M}$ and arrows are defined for all m, n elements of $\mathcal{M}$ as follows: $f_{m,n} : m \to n$ if $m \leq n$, with $f_{m,m} = id_m$ and $f_{n,m}; f_{m,o} = f_{n,o}$.*

Note that $\mathcal{C}(\mathcal{M})$ is a strict monoidal category, inheriting its structure from $\mathcal{M}$. It is a thin category (for each pair of objects there is at most one arrow) and it is strictly symmetric if $\otimes$ is commutative, yet it is not skeletal, since isomorphisms are not identities. All arrows are mono as well as epi, and 1 is the initial object, while pushouts and pullbacks do not necessarily exist.

Example 13. The corresponding category of computations $\mathcal{C}(\mathcal{M})$ for the implementation of a *multi-value Register* (Ex. 11) has sets $S \in 2^{\mathbb{N}\times\mathbb{N}^{\mathcal{R}}}$ as objects and arrows $f_{S,S'} : S \to S'$ with $S \leq S'$. We may attempt to interpret labels as arrows of $\mathcal{C}(\mathcal{M})$, which would work for the operations of the data type, i.e., `wr` and `rd`, even if the latter might be partial. For `rcv`, however, the target of each arrow would depend on the received state. We avoid considering different receive operations, and we describe them non-deterministically via the power-domain construction. In fact, the objects of $\mathbf{P}(\mathcal{C}(\mathcal{M}))$ are sets of sets of pairs, i.e., $X \subseteq 2^{\mathbb{N}\times\mathbb{N}^{\mathcal{R}}}$. Each arrow $(R, f_R) : X \to Y$ represents a computation in which a replica starting on some state $S \in X$ will end up in some state $S' \in (R, f_R)(S)$.

With the monoidal structure in place, we instantiate our development of the previous section and obtain an implementation LTS better suited for our notion of simulation. Implementation functors are of the form $\mathbb{I}\colon \mathbf{IR}(\mathcal{R}) \to \mathbf{P}(\mathcal{C}(\mathcal{M}))$, with $\mathcal{M}$ a pre-ordered monoid, and the derived replica LTS has transitions

$$\langle r, m\rangle \xrightarrow{\langle g, \overline{g}\rangle} \langle r, n\rangle$$

where m and n are elements of the monoid. We impose the further requirement that `rcv` must accept any possible state that it receives, so that $\mathbb{I}(r)$ is a submonoid of $\mathcal{M}$ and $\mathbb{I}(\texttt{rcv}_r)(m) = m \otimes \mathbb{I}(r)$ for all $r \in \mathcal{R}$ and $m \in \mathbb{I}(r)$. This implies that $\mathbb{I}(\texttt{rcv}_r) = \mathbb{I}(\texttt{rcv}_s)$ for all $r, s \in \mathcal{R}$ and that the operation behaves symmetrically, i.e. $m \otimes o \in \mathbb{I}(\texttt{rcv}_r)(m) \cap \mathbb{I}(\texttt{rcv}_r)(o)$ for all $r \in \mathcal{R}$ and $m, o \in \mathbb{I}(r)$.

Finally, with an abuse of notation, we denote as ℓ the label $\langle \ell, id_r \rangle$ and as $\langle s, o \rangle$ the label $\langle \texttt{rcv}, \overline{g} \rangle$ such that $\overline{g} : \langle s, \mathbb{I}(\iota_{r,s})(o) \rangle \to \langle r, m \otimes o \rangle$ is obtained by composing the arrows $\iota_{s,r} : \langle s, \mathbb{I}(\iota_{r,s})(o) \rangle \to \langle r, o \rangle$ and $\texttt{rcv}_r : \langle r, o \rangle \to \langle r, m \otimes o \rangle$ (noting that there is no ambiguity since $\mathbb{I}(\iota_{r,s})(o)$ is a singleton).

Example 14. We now consider the implementation functor $\mathbb{I}$ for the *multi-value Register*. We have $\mathbb{I}(r) = \{\|S\| \mid S \in 2^{\mathbb{N} \times \mathbb{N}^{\mathcal{R}}}\}$, i.e., a set containing all sets of maximal pairs. An arrow in $\mathbf{IR}(\mathcal{R})$ (i.e., a sequence of operations) is mapped by $\mathbb{I}$ to an arrow $\mathbb{I}(r) \to \mathbb{I}(r)$. For instance, $\mathbb{I}(\langle \texttt{wr(a)}, ok \rangle_r)$ is defined such that $\mathbb{I}(\langle \texttt{wr(a)}, ok \rangle_r)(S) = \{\{(\texttt{a}, \pi_2 S \triangleright r)\}\}$ for all $S \in \mathbb{I}(r)$, i.e., `wr(a)` can be performed over any state S, and this operation (deterministically) changes the state by a set containing just a pair with the written value and a dominating version vector. Analogously, $\mathbb{I}(\langle \texttt{rd}, V \rangle_r)$ is defined such that $\mathbb{I}(\langle \texttt{rd}, V \rangle_r)(S) = \{S\}$ if $\pi_1 S = V$ and $\mathbb{I}(\langle \texttt{rd}, V \rangle_r)(S) = \emptyset$ otherwise, i.e., a read operation that returns V can be performed only over a state S that contains exactly the values V, otherwise such an operation cannot occur. Finally, $\mathbb{I}(\texttt{rcv}_r)(S) = \{S \oplus S' \mid S' \in \mathbb{I}(r)\}$, i.e., a receive can augment the state S with any received state S'.

Now, the category of elements for the functor $\mathbb{I}$ introduced above generates the transition system in Fig. 5, if we disregard labels. Labels are actually recovered by the corresponding implementation LTS; in particular the label $\langle \texttt{rcv}, \overline{g} \rangle$ makes the connection between the received state and the target of the arrow, which is analogous to rule (Rcv).

5.4 Implementation correctness via simulation

We are now ready to characterise implementation correctness as a simulation relation between the context LTS and the implementation LTS for a given RDT. The starting point is what usually occurs in higher-order calculi: Since the label of a transition may be a process, the notion of simulation has to take also labels into account. Thus, our proposal is the following.

Definition 20 (Implementation correctness). *Let $\mathcal{S}$ be a specification, $\mathcal{C}_{\mathcal{S}}$ the context LTS, and $\mathcal{I}_{\mathcal{S}}$ the implementation LTS. An* implementation relation *$\mathcal{R}_{\mathcal{S}}$ is a relation between states in $\mathcal{I}_{\mathcal{S}}$ and $\mathcal{C}_{\mathcal{S}}$ such that if $(\sigma, \langle \texttt{G}, \texttt{P} \rangle) \in \mathcal{R}_{\mathcal{S}}$ then*

1. *if $\sigma \xrightarrow{\ell} \sigma'$ then $\exists \texttt{G}', \texttt{P}'$ such that $\langle \texttt{G}, \texttt{P} \rangle \xrightarrow{\ell} \langle \texttt{G}', \texttt{P}' \rangle$ and $(\sigma', \langle \texttt{G}', \texttt{P}' \rangle) \in \mathcal{R}_{\mathcal{S}}$;*
2. *if $\sigma \xrightarrow{\sigma'} \sigma''$ then $\exists \texttt{G}', \texttt{G}'', \texttt{P}', \texttt{P}''$ such that $\langle \texttt{G}, \texttt{P} \rangle \xrightarrow{\langle \texttt{G}', \texttt{P}' \rangle} \langle \texttt{G}'', \texttt{P}'' \rangle$, $(\sigma', \langle \texttt{G}', \texttt{P}' \rangle) \in \mathcal{R}_{\mathcal{S}}$, and $(\sigma'', \langle \texttt{G}'', \texttt{P}'' \rangle) \in \mathcal{R}_{\mathcal{S}}$.*

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

Given the way we distilled labels, the definition above does coincide with the notion of implementation correctness as given in [7, Definition 21]. A distinction between one- and multi-replica simulation can be recovered just by suitably restricting the context LTS, that is, item 1 above, by requiring ℓ to be arising from either a root extension or an extension, respectively.

6 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 [9] and improves [7] and similar set-theoretical characterisations, which are now made precise and recast into standard notions from the literature, 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 [7], where the action of sending will play 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 pre-sheaves and simulation, most notably in [17]. The distilling of labels is clearly reminiscent of the *contexts as labels* paradigm advanced by Leifer and Milner [12], and it would fit in its less constrained version proposed in [1]. Since this was not the main methodological issue of the paper, we adopted a presentation requiring some ingenuity.

References

1. 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)
2. 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)
3. Burckhardt, S., Gotsman, A., Yang, H.: Understanding eventual consistency. Tech. Rep. MSR-TR-2013-39, Microsoft Research (2013)
4. 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)
5. 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)

6. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. In: Bressoud, T.C., Kaashoek, M.F. (eds.) SOSP 2007. pp. 205–220. ACM (2007)
7. Gadducci, F., Melgratti, H., Roldán, C.: On the semantics and implementation of replicated data types. Science of Computer Programming **167**, 91–113 (2018)
8. 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)
9. 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)
10. 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)
11. 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)
12. 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)
13. MacLane, S., Moerdijk, I.: Sheaves in geometry and logic: A first introduction to topos theory. Springer (1992)
14. 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)
15. 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)
16. 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)
17. Sobociński, P.: Relational presheaves, change of base and weak simulation. Journal of Computer and System Sciences **81**(5), 901–910 (2015)