A denotational view of replicated data types ⋆

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

[1] Dipartimento di Informatica, Università di Pisa, Italia
[2] Departamento de Computación, FCEyN, Universidad de Buenos Aires, Argentina
[3] CONICET

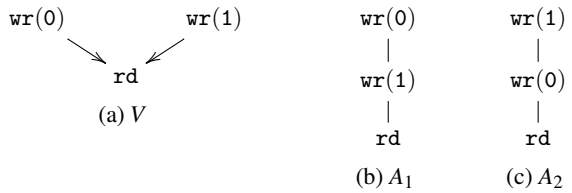
Abstract. "Weak consistency" refers to a family of properties concerning the state of a distributed system. One of the key issues in their description is the way in which systems are specified. In this regard, a major advance is represented by the introduction of Replicated Data Types (RDTs), in which the meaning of operators is given in terms of two relations, namely, visibility and arbitration. Concretely, a data type operation is defined as a function that maps visibility and arbitration into a return value. In this paper we recast such standard approaches into a denotational framework in which a data type is seen as a function that maps visibility into admissible arbitrations. This characterisation provides a more abstract view of RDTs that (i) highlights some of the implicit assumptions shared in operational approaches to specification; (ii) accommodates underspecification and refinement; (iii) enables a categorical presentation of RDT and the development of composition operators for specifications.

1 Introduction

Distributed systems replicate their state over different nodes in order to satisfy several non-functional requirements, such as performance, availability, and reliability. It then becomes crucial to keep a consistent view of the replicated data. However, this is a challenging task because consistency is in conflict with two common requirements of distributed applications: *availability* (every request is eventually executed) and tolerance to network *partitions* (the system operates even in the presence of failures that prevent communication among components). In fact, it is impossible for a system to simultaneously achieve strong Consistency, Availability and Partition tolerance [6]. Since many domains cannot renounce to availability and network partitions, developers need to cope with weaker notions of consistency by allowing, e.g., replicas to (temporarily) exhibit some discrepancies, as long as they eventually converge to the same state.

This setting challenges the way in which data are specified: states, state transitions and return values should account for the different views that a data item may simultaneously have. Consider a data type `Register` corresponding to a memory cell that is read and updated by using, respectively, operations `rd` and `wr`. In a replicated scenario, the value obtained when reading a register after two concurrent updates $\texttt{wr}(0)$ and $\texttt{wr}(1)$

⋆ The first author has been partially supported by CONICET International Cooperation Grant 995/15. Research partially supported by UBACyT project 2014-2017 20020130200092BA and CONICET project PIP 11220130100148CO.



(a) V (b) A_1 (c) A_2

Fig. 1. A scenario for the replicated data type `Register`

(i.e., updates taking place over different replicas) is affected by the way in which updates propagate among the different replicas: it is perfectly possible that the result of the read is (i) undefined (when the read is performed over a third replica that has not received any of the updates), (ii) 0 or (iii) 1. Basically, the return value depends on the updates that are seen by that read operation. Choosing the return value is straightforward when a read sees just one update. This is less so if a read is performed over a replica that knows both updates, for allowing all replicas to (consistently) pick one of the available values. A common strategy for registers is that the *last-write wins*, i.e., the last update should be chosen when several concurrent updates are observed. This strategy implicitly assumes that all events in a system can be arranged in a total order. Several recent approaches focus on the operational specification of replicated data types [12, 4, 2, 3, 7, 5, 14, 8]. Usually, the specification describes the meaning of an operation in terms of two different relations among events: *visibility*, which explains the causes for each result, and *arbitration*, which totally orders events. Consider the visibility relation V in Fig. 1a and the arbitrations A_1 and A_2 in Fig. 1b and Fig. 1c, respectively. The meaning of `rd` is defined such that $\mathtt{rd}(V,A_1) = 1$ and $\mathtt{rd}(V,A_2) = 0$. We remark that operational approaches require specifications to be *functional*, i.e., for every operation, visibility and arbitration relation, there exists exactly one return value. In this way operational specifications commit to concrete policies for resolving conflicts.

This work aims at putting on firm grounds the operational approaches for RDTs by giving them a purely functional description and, eventually, a categorical one. In our view, RDTs are functions that map visibility graphs (i.e., configurations) into sets of admissible arbitrations, i.e., all executions that generate a particular configuration. In this setting, a configuration mapped to an empty set of admissible arbitrations stands for an unreachable configuration. We rely on such an abstract view of RDTs to highlight some of the implicit assumptions shared by most of the operational approaches. In particular, we characterise operational approaches, such as [12, 4], as those specifications that satisfy three properties: besides the evident requirement of being *functional* (i.e., deterministic and total), they must be *coherent* (i.e., larger states are explained as the composition of smaller ones), and *saturated* (e.g., an unobserved operation can be arbitrated in any position, even before the events that it sees). We show this inclusion to be strict and discuss some interesting cases that do not fall in this class. Moreover, we show that functional characterisation elegantly accounts for underspecification and refinement, which are standard notions in data type specification.

Then, we develop a categorical presentation for specifications. We focus on coherent specifications and show that there is a one-to-one correspondence between coherent specifications and a particular class of functors from the category $I(\mathcal{L})$ of labelled directed acyclic graphs and injective *past-reflecting morphism* (which are the dual notion of tp-morphisms [9]) to the category $\mathcal{P}(\mathcal{L})$ of sets of paths and path-set morphisms preserving the initial object. As it is standard from classical results on algebraic specification theory, pullbacks and (a weak form of) pushouts in $I(\mathcal{L})$ provide basic operators for composing specifications, and thus our functorial presentation is the first step towards a denotational semantics of RDTs (see e.g. [1] and the references therein).

The paper has the following structure. Section 2 introduces the basic definitions concerning labelled directed acyclic graphs. Section 3 discusses our functional mechanism for the presentation of Replicated Data Types. Section 4 compares our proposal with the classical operational one [2]. Section 5 illustrates a categorical characterisation for our proposal. Finally, in the closing section we draw some conclusions and highlight further developments.

2 Labelled Directed Acyclic Graphs

In this section we recall the basics of labelled directed acyclic graphs, which are used for our description of replicated data types. We rely on countable sets $\mathcal{E}$ of events $\mathtt{e}, \mathtt{e}', \ldots, \mathtt{e}_1, \ldots$ and $\mathcal{L}$ of labels $\ell, \ell', \ldots, \ell_1, \ldots$

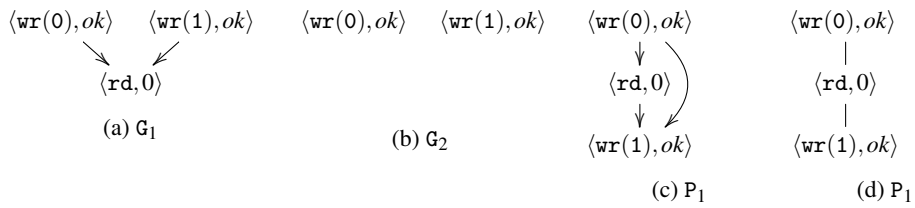
Definition 1 (Labelled Directed Acyclic Graph). *A* Labelled Directed Acyclic Graph *(*LDAG*)* over *a set of labels $\mathcal{L}$ is a triple $\mathtt{G} = \langle \mathcal{E}_\mathtt{G}, \prec_\mathtt{G}, \lambda_\mathtt{G} \rangle$ such that $\mathcal{E}_\mathtt{G}$ is a set of events, $\prec_\mathtt{G} \subseteq \mathcal{E}_\mathtt{G} \times \mathcal{E}_\mathtt{G}$ is a binary relation whose transitive closure is a strict partial order, and $\lambda_\mathtt{G} : \mathcal{E}_\mathtt{G} \to \mathcal{L}$ is a labeling function. An* LDAG $\mathtt{G}$ *is a* path *if $\prec_\mathtt{G}$ is a strict total order.*

We write $\mathbb{G}(\mathcal{L})$ and $\mathbb{P}(\mathcal{L})$ to respectively denote the sets of all LDAGs and paths over $\mathcal{L}$. We use $\mathtt{G}$ to range over $\mathbb{G}(\mathcal{L})$ and $\mathtt{P}$ to range over $\mathbb{P}(\mathcal{L})$. Moreover, we write $<_\mathtt{P}$ instead of $\prec_\mathtt{P}$ to make evident that paths are total orders. We say that $\mathtt{P}$ is a path over $\mathcal{E}$ if $\mathcal{E}_\mathtt{P} = \mathcal{E}$ and write $\mathbb{P}(\mathcal{E}, \lambda)$ for $\{\mathtt{P} \mid \mathtt{P} \text{ is a path over } \mathcal{E} \text{ and } \lambda_\mathtt{P} = \lambda\}$. We usually omit the subscript $\mathtt{G}$ (or $\mathtt{P}$) when referring to the elements of $\mathtt{G}$ (of $\mathtt{P}$, respectively) when no confusion arises. We write ε for the empty LDAG, i.e., such that $\mathcal{E}_\varepsilon = \emptyset$.

Definition 2 (Morphism). *An* LDAG *morphism* $\mathtt{f}$ *from* $\mathtt{G}$ *to* $\mathtt{G}'$*, written* $\mathtt{f} : \mathtt{G} \to \mathtt{G}'$*, is a mapping* $\mathtt{f} : \mathcal{E}_\mathtt{G} \to \mathcal{E}_{\mathtt{G}'}$ *such that* $\lambda_\mathtt{G} = \mathtt{f}; \lambda_{\mathtt{G}'}$ *and* $\mathtt{e} \prec_\mathtt{G} \mathtt{e}'$ *implies* $\mathtt{f}(\mathtt{e}) \prec_{\mathtt{G}'} \mathtt{f}(\mathtt{e}')$.

Hereafter we implicitly consider LDAGs up-to isomorphism, i.e., related by a bijective function that preserves and reflects the underlying relation.

Example 1. Consider the set $\mathcal{L} = \{\langle \mathtt{rd}, 0\rangle, \langle \mathtt{rd}, 1\rangle, \langle \mathtt{wr}(0), ok\rangle, \langle \mathtt{wr}(1), ok\rangle\}$ of labels describing the operations of a 1-bit register. Each label is a tuple $\langle op, rv\rangle$ where *op* denotes an operation and *rv* its return value. For homogeneity, we associate the return value *ok* to every write operation. Now, take the LDAG over $\mathcal{L}$ defined as $\mathtt{G}_1 = \langle \{\mathtt{e}_1, \mathtt{e}_2, \mathtt{e}_3\}, \prec, \lambda\rangle$ where $\prec = \{(\mathtt{e}_1, \mathtt{e}_3), (\mathtt{e}_2, \mathtt{e}_3)\}$ and λ is such that $\lambda(\mathtt{e}_1) = \langle \mathtt{wr}(0), \mathtt{ok}\rangle$, $\lambda(\mathtt{e}_2) = \langle \mathtt{wr}(1), \mathtt{ok}\rangle$, $\lambda(\mathtt{e}_3) = \langle \mathtt{rd}, 0\rangle$. A graphical representation of $\mathtt{G}_1$ is in Fig. 2a.



(a) $\mathtt{G}_1$ (b) $\mathtt{G}_2$ (c) $\mathtt{P}_1$ (d) $\mathtt{P}_1$

Fig. 2. Two simple LDAGs and two paths.

Since we consider LDAGs up-to isomorphism, we do not depict events and write instead the corresponding labels when no confusion arises. $\mathtt{G}_2$ is an LDAG where $\prec_{\mathtt{G}_2}$ is empty. Neither $\mathtt{G}_1$ nor $\mathtt{G}_2$ is a path, because they are not total orders. $\mathtt{P}_1$ in Fig. 2c is an LDAG that is also a path. Hereinafter we use undirected arrows when depicting paths and avoid drawing transitions that are obtained by transitivity, as shown in Fig. 2d. All LDAGs in Fig. 2 belong to $\mathbb{G}(\mathcal{L})$, but only $\mathtt{P}_1$ is in $\mathbb{P}(\mathcal{L})$.

2.1 LDAG operations

We now present a few operations on LDAGs, which will be used in the following sections. We start by introducing some notation for binary relations. We write $\textsc{Id}$ for the identity relation over events and $\preccurlyeq$ for $\prec \cup \textsc{Id}$. We write $- \prec \mathtt{e}$ (and similarly $- \preccurlyeq \mathtt{e}$) for the preimage of $\mathtt{e}$, i.e., $- \prec \mathtt{e} = \{\mathtt{e}' \mid \mathtt{e}' \prec \mathtt{e}\}$. We use $\prec|_{\mathcal{E}}$ for the restriction of $\prec$ to elements in $\mathcal{E}$, i.e. $\prec|_{\mathcal{E}} = \prec \cap (\mathcal{E} \times \mathcal{E})$. Analogously, $\lambda|_{\mathcal{E}}$ is the domain restriction of λ to the elements in $\mathcal{E}$. We write $\mathcal{E}_\top$ for the extension of the set $\mathcal{E}$ with a fresh element, i.e., $\mathcal{E}_\top = \mathcal{E} \cup \{\top\}$ such that $\top \notin \mathcal{E}$.

Definition 3 (Restriction and Extension). *Let* $\mathtt{G} = \langle \mathcal{E}, \prec, \lambda \rangle$ *and* $\mathcal{E}' \subseteq \mathcal{E}$. *We define*

- $\mathtt{G}|_{\mathcal{E}'} = \langle \mathcal{E}', \prec|_{\mathcal{E}'}, \lambda|_{\mathcal{E}'} \rangle$ *as the* restriction of $\mathtt{G}$ to $\mathcal{E}'$*;*
- $\mathtt{G}^{\ell}_{\mathcal{E}'} = \langle \mathcal{E}_\top, \prec \cup (\mathcal{E}' \times \{\top\}), \lambda[\top \mapsto \ell] \rangle$ *as the* extension of $\mathtt{G}$ over $\mathcal{E}'$ with ℓ.

Restriction obviously lifts to sets $\mathcal{X}$ of LDAGs, i.e., $\mathcal{X}|_{\mathcal{E}} = \{\, \mathtt{G}|_{\mathcal{E}} \mid \mathtt{G} \in \mathcal{X} \}$. We omit the subscript $\mathcal{E}'$ in $\mathtt{G}^{\ell}_{\mathcal{E}'}$ when $\mathcal{E}' = \mathcal{E}$.

Example 2. Consider the LDAGs $\mathtt{G}_1$ and $\mathtt{G}_2$ depicted in Fig. 2a and Fig. 2b, respectively. Then, $\mathtt{G}_2 = \mathtt{G}_1|_{-\prec \mathtt{e}_3}$ and $\mathtt{G}_1 = \mathtt{G}_2{}^{\langle \mathtt{rd},0 \rangle}$.

The following operator allows for the combination of several paths and plays a central rol in our characterisation of replicated data types.

Definition 4 (Product). *Let* $\mathcal{X} = \{\langle \mathcal{E}_i, <_i, \lambda_i \rangle\}_i$ *be a set of paths. The product of* $\mathcal{X}$ *is*

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

Intuitively, the product of paths is analogous to the synchronous product of transition systems, in which common elements are identified and the remaining ones can be freely interleaved, as long as the original orders are respected.

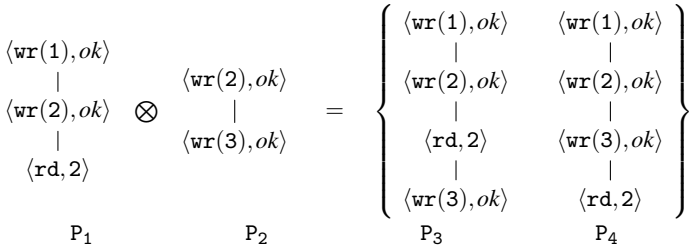


Fig. 3. Product between two paths.

Example 3. Consider the paths $\mathtt{P}_1$ and $\mathtt{P}_2$ in Fig. 3, and assume that they share the event labelled $\langle \mathtt{wr}(2), ok \rangle$. Their product has two paths $\mathtt{P}_3$ and $\mathtt{P}_4$, each of them contains the elements of $\mathtt{P}_1$ and $\mathtt{P}_2$ and preserves the relative order of the elements in the original paths. We remark that the product is empty when the paths have incompatible orders. For instance, $\mathtt{P}_3 \otimes \mathtt{P}_4 = \emptyset$.

It is straightforward to show that $\otimes$ is associative and commutative. Hence, we freely use $\otimes$ over sets of sets of paths.

3 Specifications

We introduce our notion of specification and applies it to some well-known data types.

Definition 5 (Specification). *A* specification $\mathcal{S}$ *is a function* $\mathcal{S} : \mathbb{G}(\mathcal{L}) \to 2^{\mathbb{P}(\mathcal{L})}$ *such that* $\mathcal{S}(\varepsilon) = \{\varepsilon\}$ *and* $\forall \mathtt{G}.\ \mathcal{S}(\mathtt{G}) \subseteq 2^{\mathbb{P}(\mathcal{E}_\mathtt{G}, \lambda_\mathtt{G})}$.

A specification $\mathcal{S}$ maps an LDAG (i.e., a visibility relation) to a set of paths (i.e., its admissible arbitrations). Note that $\mathtt{P} \in \mathcal{S}(\mathtt{G})$ is a path over $\mathcal{E}_\mathtt{G}$, and hence a total order of the events in $\mathtt{G}$. However, we do not require $\mathtt{P}$ to be a topological ordering of $\mathtt{G}$, i.e., $\prec_\mathtt{G} \subseteq <_\mathtt{P}$ may not hold. Although some specification approaches consider only arbitrations that include visibility [7, 5], our definition accommodates also presentations, such as [4, 2], in which arbitrations may not preserve visibility. We focus later on in a few subclasses, such as coherent specifications, in order to establish a precise correspondence with replicated data types. We also remark that it could be the case that $\mathcal{S}(\mathtt{G}) = \emptyset$, which means that $\mathcal{S}$ forbids the configuration $\mathtt{G}$ (more details in Ex. 4 below). For technical convenience, we impose $\mathcal{S}(\varepsilon) = \{\varepsilon\}$ and disallow $\mathcal{S}(\varepsilon) = \emptyset$: $\mathcal{S}$ cannot forbid the empty configuration, which denotes the initial state of a data type.

We now illustrate the specification of some well-known replicated data types.

Example 4 (Counter). The data type `Counter` provides operations for incrementing and reading an integer register with initial value 0. A read operation returns the number of increments seen by that read. An increment is always successful and returns the

$$\mathcal{S}_{Ctr}\left(\begin{array}{c}\langle \texttt{inc}, ok\rangle \\ \downarrow \\ \langle \texttt{rd}, 1\rangle\end{array}\right) = \left\{\begin{array}{cc}\langle \texttt{inc}, ok\rangle & \langle \texttt{rd}, 1\rangle \\ | & | \\ \langle \texttt{rd}, 1\rangle & \langle \texttt{inc}, ok\rangle\end{array}\right\} \qquad \mathcal{S}_{Ctr}\left(\begin{array}{c}\langle \texttt{inc}, ok\rangle \\ \downarrow \\ \langle \texttt{rd}, 0\rangle\end{array}\right) = \emptyset$$

(a) (b)

Fig. 4. Counter specification.

value ok. Formally, we consider the set of labels $\mathcal{L} = \{\langle \texttt{inc}, ok\rangle\} \cup (\{\texttt{rd}\} \times \mathbb{N})$. Then, a `Counter` is specified by $\mathcal{S}_{Ctr}$ defined such that

$$\texttt{P} \in \mathcal{S}_{Ctr}(\texttt{G}) \text{ if } \forall \texttt{e} \in \mathcal{E}_{\texttt{G}}.\lambda(\texttt{e}) = \langle \texttt{rd}, k\rangle \text{ implies } k = \#\{\texttt{e}' \mid \texttt{e}' \prec_{\texttt{G}} \texttt{e} \text{ and } \lambda(\texttt{e}') = \langle \texttt{inc}, ok\rangle\}$$

A visibility graph G has admissible arbitrations (i.e., $\mathcal{S}_{Ctr}(\texttt{G}) \neq \emptyset$) only when each event e in G labelled by `rd` has a return value k that matches the number of increments anteceding e in G. We illustrate two cases for the definition of $\mathcal{S}_{Ctr}$ in Fig. 4. While the configuration in Fig. 4a has admissible arbitrations, the one in Fig. 4b has not, because the unique event labelled by `rd` returns 0 when it is actually preceded by an observed increment. In other words, an execution is not allowed to generate such a visibility graph. We remark that $\mathcal{S}_{Ctr}$ does not impose any constraint on the ordering $<_{\texttt{P}}$.

In fact, a path $\texttt{P} \in \mathcal{S}_{Ctr}(\texttt{G})$ does not need to be a topological ordering of G as, for instance, the rightmost path in the set of Fig. 4a.

Example 5 (Last-write-wins Register). A `Register` stores a value that can be read and updated. We assume that the initial value of a register is undefined. We take $\mathcal{L} = \{\langle \texttt{wr}(k), ok\rangle \mid k \in \mathbb{N}\} \cup (\{\texttt{rd}\} \times \mathbb{N} \cup \{\bot\})$ as the set of labels. The specification $\mathcal{S}_{lwwR}$ gives the semantics of a register that adopts the last-write-wins strategy.

$$\texttt{P} \in \mathcal{S}_{lwwR}(\texttt{G}) \text{ if } \forall \texttt{e} \in \mathcal{E}_{\texttt{G}}. \begin{cases} \lambda(\texttt{e}) = \langle \texttt{rd}, \bot\rangle \text{ implies } \forall \texttt{e}' \prec_{\texttt{G}} \texttt{e}.\forall k.\lambda(\texttt{e}') \neq \langle \texttt{wr}(k), ok\rangle \\ \lambda(\texttt{e}) = \langle \texttt{rd}, k\rangle \text{ implies } \exists \texttt{e}' \prec_{\texttt{G}} \texttt{e}.\lambda(\texttt{e}') = \langle \texttt{wr}(k), ok\rangle \text{ and} \\ \quad \forall \texttt{e}'' \prec_{\texttt{G}} \texttt{e}.\ \texttt{e}' <_{\texttt{P}} \texttt{e}'' \text{ implies } \forall k'.\lambda(\texttt{e}'') \neq \langle \texttt{wr}(k'), ok\rangle \end{cases}$$

An LDAG G has admissible arbitrations only when each event associated with a read operation returns a previously written value. As per the first condition above, a read operation returns the undefined value $\bot$ when it does not see any write. By the second condition, a read e returns a natural number k when it sees an operation $\texttt{e}'$ that writes that value k. In such case, any admissible arbitration P must order $\texttt{e}'$ as the greatest (accordingly to $<_{\texttt{P}}$) of all write operations seen by e.

Example 6 (Generic Register). We now define a `Generic Register` that does no commit to a particular strategy for resolving conflicts. We specify this type as follows

$$\texttt{P} \in \mathcal{S}_{gR}(\texttt{G}) \text{ if } \forall \texttt{e} \in \mathcal{E}_{\texttt{G}}. \begin{cases} \lambda(\texttt{e}) = \langle \texttt{rd}, \bot\rangle \text{ implies } \forall \texttt{e}' \prec_{\texttt{G}} \texttt{e}.\forall k.\lambda(\texttt{e}') \neq \langle \texttt{wr}(k), ok\rangle \\ \lambda(\texttt{e}) = \langle \texttt{rd}, k\rangle \text{ implies } \exists \texttt{e}' \prec_{\texttt{G}} \texttt{e}.\lambda(\texttt{e}') = \langle \texttt{wr}(k), ok\rangle \text{ and} \\ \quad \forall \texttt{e}''.\lambda(\texttt{e}'') = \langle \texttt{rd}, k''\rangle \text{ and } - \prec_{\texttt{G}} \texttt{e} = - \prec_{\texttt{G}} \texttt{e}'' \text{ implies } k = k'' \end{cases}$$

As in Ex. 5, the return value of a read corresponds to a written value seen by that read, but the specification does not determine which value should be chosen. We require instead that all read operations with the same causes (i.e., $- \prec_{\mathtt{G}} \mathtt{e} = - \prec_{\mathtt{G}} \mathtt{e}'$) have the same result. Since this condition is satisfied by any admissible configuration $\mathtt{G}$, it ensures convergence. The fact that convergence is explicitly required contrasts with approaches like [4, 2], where on the contrary convergence is ensured automatically by considering only deterministic specifications. We remark that for the deterministic cases, e.g., Ex. 4 and Ex. 5, we do not need to explicitly require convergence.

3.1 Refinement

Refinement is a standard approach in data type specification, which allows for a hierarchical organisation that goes from abstract descriptions to concrete implementations. The main benefit of refinement relies on the fact that applications can be developed and reasoned about in terms of abstract types, which hide implementation details and leave some freedom for the implementation. Consider the specification $\mathcal{S}_{gR}$ of the `Generic Register` introduced in Ex. 6, which only requires a policy for conflict resolution that ensures convergence. On the contrary, the specification $\mathcal{S}_{lwwR}$ in Ex. 5 explicitly states that concurrent updates must be resolved by adopting the last-write-wins policy. Since the latter policy ensures convergence, we would like to think about $\mathcal{S}_{lwwR}$ as a refinement of $\mathcal{S}_{gR}$. We characterise refinement in our setting as follows.

Definition 6 (Refinement). *Let $\mathcal{S}_1, \mathcal{S}_2$ be specifications. We say that $\mathcal{S}_1$ refines $\mathcal{S}_2$ and we write $\mathcal{S}_1 \sqsubseteq \mathcal{S}_2$ if $\forall \mathtt{G}.\ \mathcal{S}_1(\mathtt{G}) \subseteq \mathcal{S}_2(\mathtt{G})$.*

Example 7. It can be easily checked that $\mathtt{P} \in \mathcal{S}_{lwwR}(\mathtt{G})$ implies $\mathtt{P} \in \mathcal{S}_{gR}(\mathtt{G})$ for any $\mathtt{G}$. Consequently, $\mathcal{S}_{lwwR}$ is a refinement of $\mathcal{S}_{gR}$.

Example 8. Consider the data type `Set`, which provides (among others) the operations `add`, `rem` and `lookup` for respectively 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 of elements (see [13] for a detailed discussion). We illustrate two possible alternatives by considering the execution scenario depicted in Fig. 5. A reasonable semantics for `lookup` over $\mathtt{G}$ and $\mathtt{P}$ would fix the result $\mathtt{V}$ as either $\emptyset$ or $\{1\}$. In fact, under the last-write-wins policy, the specification prescribes that `lookup` returns $\{1\}$ in this scenario. Differently, the strategy of 2P-Sets[4] establishes that the result is $\emptyset$.

The following definition provides a specification for an abstract data type `Set` that allows (among others) any of the above policies.

$$\mathtt{P} \in \mathcal{S}_{Set}(\mathtt{G}) \text{ if } \forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}}.\lambda(\mathtt{e}) = \langle \mathtt{lookup}, \mathtt{V} \rangle \text{ implies } B_{\mathtt{e}} \subseteq \mathtt{V} \subseteq A_{\mathtt{e}} \text{ and } \mathtt{Conv}_{\mathtt{e},\mathtt{V}}$$

where

$$A_{\mathtt{e}} = \{k \mid \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e} \text{ and } \lambda(\mathtt{e}') = \langle \mathtt{add}(k), ok \rangle\}$$
$$B_{\mathtt{e}} = A_{\mathtt{e}} \setminus \{k \mid \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e} \text{ and } \lambda(\mathtt{e}') = \langle \mathtt{rem}(k), ok \rangle\}$$
$$\mathtt{Conv}_{\mathtt{e},\mathtt{V}} = \forall \mathtt{e}'.\lambda(\mathtt{e}') = \langle \mathtt{lookup}, \mathtt{V}' \rangle \text{ and } - \prec_{\mathtt{G}} \mathtt{e} = - \prec_{\mathtt{G}} \mathtt{e}' \text{ implies } \mathtt{V} = \mathtt{V}'$$

[4] In 2P-Sets, additions of elements that have been previously removed have no effect

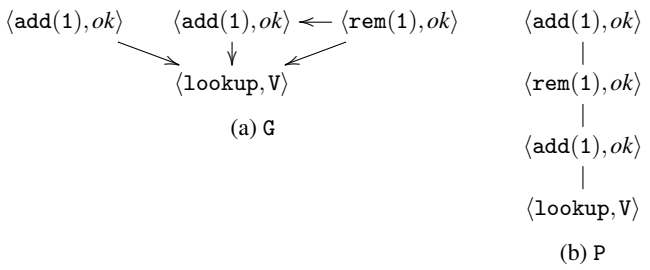


Fig. 5. A scenario for the replicated data type `Set`

The set $A_{\mathtt{e}}$ contains the elements added to (and possibly removed from) the set seen by e while $B_{\mathtt{e}}$ contains those elements for which e sees no removal. Thus, the condition $B_{\mathtt{e}} \subseteq \mathtt{V} \subseteq A_{\mathtt{e}}$ states that `lookup` returns a set that contains at least all the elements added but not removed (i.e., in $B_{\mathtt{e}}$). However, the return value V may contain elements that have been added and removed (the choice is left unspecified). Condition `Conv` ensures convergence, similarly to the specification of $\mathcal{S}_{gR}$ in Ex. 6.

Then, a concrete resolution policy such as 2P-Sets can be specified as follows

$$\mathtt{P} \in \mathcal{S}_{\textit{2P-Sets}}(\mathtt{G}) \text{ if } \forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}}.\lambda(\mathtt{e}) = \langle \mathtt{lookup}, \mathtt{V} \rangle \text{ implies } \mathtt{V} = B_{\mathtt{e}}$$

Clearly, $\mathcal{S}_{\textit{2P-Sets}}$ is a refinement of $\mathcal{S}_{\textit{Sets}}$. Other policies can be specified analogously.

3.2 Classes of specifications

We now discuss two properties of specifications. Firstly, we look at specifications for which the behaviour of larger computations matches that of their shorter prefixes.

Definition 7 (Past-Coherent Specification). *Let $\mathcal{S}$ be a specification. We say that $\mathcal{S}$ is past-coherent (briefly, coherent) if*

$$\forall \mathtt{G}.\ \mathcal{S}(\mathtt{G}) = \bigotimes_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} \mathcal{S}(\mathtt{G}|_{-\preccurlyeq \mathtt{e}})$$

Note that coherence implies that $\mathcal{S}(\mathtt{G})|_{-\prec \mathtt{e}} \subseteq \mathcal{S}(\mathtt{G}|_{-\prec \mathtt{e}})$. Intuitively, sub-paths are obtained from the interleaving of the paths belonging to the associated sub-specifications.

Example 9. The specifications in Ex. 4, Ex. 5 and Ex. 6 are all coherent, because their definitions are in terms of restrictions of the LDAGs. Now consider the specification $\mathcal{S}$ defined such that the equalities in Fig. 6 hold. $\mathcal{S}$ is not coherent because the arbitrations for the LDAG in Fig. 6b should contain all the interleavings for the paths associated with its sub-configurations, as depicted in Fig. 6a. Instead, note that the arbitration of $\langle \mathtt{o}_2, v_2 \rangle$ before $\langle \mathtt{o}_1, v_1 \rangle$ in the leftmost path on Fig. 6c would not hinder coherence by itself, even if it is not allowed by the sub-configuration in Fig. 6b.

$$\mathcal{S}\left(\ \langle \mathtt{o}_1, v_1\rangle\ \right) = \left\{\ \langle \mathtt{o}_1, v_1\rangle\ \right\}$$
$$\mathcal{S}\left(\ \langle \mathtt{o}_2, v_2\rangle\ \right) = \left\{\ \langle \mathtt{o}_2, v_2\rangle\ \right\}$$

(a)

$$\mathcal{S}\left(\ \langle \mathtt{o}_1, v_1\rangle \quad \langle \mathtt{o}_2, v_2\rangle\ \right) = \left\{ \begin{array}{c} \langle \mathtt{o}_1, v_1\rangle \\ | \\ \langle \mathtt{o}_2, v_2\rangle \end{array} \right\}$$

(b)

$$\mathcal{S}\left(\begin{array}{ccc} \langle \mathtt{o}_1, v_1\rangle & & \langle \mathtt{o}_2, v_2\rangle \\ & \searrow \quad \swarrow & \\ & \langle \mathtt{o}_3, v_3\rangle & \end{array} \right) = \left\{ \begin{array}{cc} \langle \mathtt{o}_1, v_1\rangle & \langle \mathtt{o}_2, v_2\rangle \\ | & | \\ \langle \mathtt{o}_2, v_2\rangle & \langle \mathtt{o}_1, v_1\rangle \\ | & | \\ \langle \mathtt{o}_3, v_3\rangle & \langle \mathtt{o}_3, v_3\rangle \end{array} \right\}$$

(c)

Fig. 6. A non-coherent specification.

A second class of specifications is concerned with saturation. Intuitively, a saturated specification allows every top element on the visibility to be arbitrated in any position. We first introduce the notion of saturation for a path.

Definition 8 (Path Saturation). *Let* $\mathtt{P}$ *be a path and* ℓ *a label. We write* $\mathtt{sat}(\mathtt{P}, \ell)$ *for the set of paths obtained by saturating* $\mathtt{P}$ *with respect to* ℓ*, defined as follows*

$$\mathtt{sat}(\mathtt{P}, \ell) = \{\mathtt{Q} \mid \mathtt{Q} \in \mathbb{P}(\mathcal{E}_{\mathtt{p}^\ell}, \lambda_{\mathtt{p}^\ell}) \text{ and } \mathtt{Q}|_{\mathcal{E}_\mathtt{P}} = \mathtt{P}\}$$

A path $\mathtt{P}$ saturated with a label ℓ generates the set of all paths obtained by placing a new event labelled by ℓ in any position within $\mathtt{P}$. A saturated specification thus extends a computation by adding a new operation that can be arbitrated in any position.

Definition 9 (Saturated Specification). *Let* $\mathcal{S}$ *be a specification. We say that* $\mathcal{S}$ *is saturated if*

$$\forall \langle \mathtt{G}, \mathtt{P}\rangle, \ell.\ \mathtt{P} \in \mathcal{S}(\mathtt{G}^\ell)\Big|_{\mathcal{E}_\mathtt{G}} \text{ implies } \mathtt{sat}(\mathtt{P}, \ell) \subseteq \mathcal{S}(\mathtt{G}^\ell)$$

Example 10. The specifications in Ex. 4, Ex. 5 and Ex. 6 are all saturated because a new event $\mathtt{e}$ can be arbitrated in any position. In fact, the specifications in Ex. 4 and Ex. 6 do not use any information about arbitration, while the specification in Ex. 5 constrains arbitrations only for events that are not maximal. Fig. 7 shows a specification that is not saturated because it does not allow to arbitrate the top event (the one labelled $\langle \mathtt{rd}, 1\rangle$) as the first operation in the path. In a saturated specification, the equality in Fig. 4a should hold. We remark that the specification is coherent although it is not saturated.

4 Replicated Data Type

In this section we show that our proposal can be considered as (and it is actually more general than) a model for the operational description of RDTs as given in [4, 2]. We start

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

Fig. 7. A non-saturated specification

by recasting the original definition of RDT (as given in [2, Def. 4.5]) in terms of LDAGs. As hinted in the introduction, the meaning of each operation of an RDT is specified in terms of a context, written $\mathtt{C}$, which is a pair $\langle \mathtt{G}, \mathtt{P}\rangle$ such that $\mathtt{P} \in \mathbb{P}(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}})$. We write $\mathbb{C}(\mathcal{L})$ for the set of contexts over $\mathcal{L}$, and fix a set O of operations and a set $\mathcal{V}$ of values. Then, the operational description of RDTs in [4, 2] can be formulated as follows.

Definition 10 (Replicated Data Type). *A* Replicated Data Type *(*RDT*) is a function* $\mathcal{F} : O \times \mathbb{C}(O) \to \mathcal{V}$.

In words, for any visibility graph $\mathtt{G}$ and arbitration $\mathtt{P}$, the specification $\mathcal{F}$ indicates the result of executing the operation $\mathtt{op}$ over $\mathtt{G}$ and $\mathtt{P}$, which is $\mathcal{F}(\mathtt{op}, \langle \mathtt{G}, \mathtt{P}\rangle)$.

Example 11. The data type `Counter` introduced in Ex. 4 is formally specified in [4, 2] as follows

$$\mathcal{F}_{ctr}(\mathtt{inc}, \langle \mathtt{G}, \mathtt{P}\rangle) = ok$$
$$\mathcal{F}_{ctr}(\mathtt{rd}, \langle \mathtt{G}, \mathtt{P}\rangle) = \#\{\mathtt{e} \mid \mathtt{e} \in \mathtt{G} \text{ and } \lambda(\mathtt{e}) = \mathtt{inc}\}$$

Given a context $\langle \mathtt{G}, \mathtt{P}\rangle$ in $\mathbb{C}(O \times \mathcal{V})$, we may check whether the value associated with each operation matches the definition of a particular RDT. This notion is known as *return value consistency* [2, Def. 4.8]. In order to relate contexts with and without return values, we use the following notation: given $\mathtt{G} \in \mathbb{G}(O \times \mathcal{V})$, by $\overline{\mathtt{G}} \in \mathbb{G}(O)$ we denote the LDAG obtained by projecting the labels of $\mathtt{G}$ in the obvious way.

Definition 11 (Return Value Consistent). *Let $\mathcal{F}$ be an* RDT *and $\langle \mathtt{G}, \mathtt{P}\rangle \in \mathbb{C}(O \times \mathcal{V})$ a context. We say that $\mathcal{F}$ is* Return Value Consistent *(*RVAL*) over $\mathtt{G}$ and $\mathtt{P}$ and we write* $\mathrm{RVAL}(\mathcal{F}, \mathtt{G}, \mathtt{P})$ *if* $\forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}}.\lambda(\mathtt{e}) = \langle \mathtt{op}, v\rangle$ *implies* $\mathcal{F}(\mathtt{op}, \overline{\mathtt{G}}|_{-\prec \mathtt{e}}, \overline{\mathtt{P}}|_{-\prec \mathtt{e}}) = v$. *Moreover, we define*

$$\mathrm{PRVAL}(\mathcal{F}, \mathtt{G}) = \{\mathtt{P} \mid \mathrm{RVAL}(\mathcal{F}, \mathtt{G}, \mathtt{P})\}$$

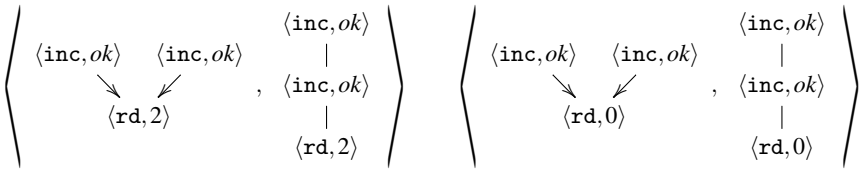
Example 12. Consider the RDT $\mathcal{F}_{ctr}$ introduced in Ex. 11. The context in Fig. 8a is RVAL consistent while the one in Fig. 8b is not because $\mathcal{F}_{ctr}$ requires `rd` to return the number of `inc` operations seen by that read, which in this case should be 2.

The following result states that return value consistent paths are all coherent, in the sense that they match the behaviour allowed for any shorter configuration.

Lemma 1. *Let $\mathcal{F}$ be an* RDT *and $\mathtt{G}$ an* LDAG. *Then*

$$\mathrm{PRVAL}(\mathcal{F}, \mathtt{G}) = \bigotimes_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} \mathrm{PRVAL}(\mathcal{F}, \mathtt{G}|_{-\preccurlyeq \mathtt{e}}).$$

As for coherent specifications, the property $\mathrm{PRVAL}(\mathcal{F}, \mathtt{G})|_{-\prec \mathtt{e}} \subseteq \mathrm{PRVAL}(\mathcal{F}, \mathtt{G}|_{-\prec \mathtt{e}})$ also holds for return value consistent paths.



(a) Consistent. (b) Non consistent.

Fig. 8. RVAL consistency for $\mathcal{F}_{ctr}$.

4.1 Deterministic specifications

We now focus on the relation between our notion of specification, as introduced in Definition 5, and the operational description of RDTs, as introduced in [4, 2] and formalised in Definition 10 in terms of LDAGs. Specifically, we characterise a proper subclass of specifications that precisely correspond to RDTs.

For this section we restrict our attention to specifications over the set of labels $O \times \mathcal{V}$, i.e., $\mathcal{S} : \mathbb{G}(O \times \mathcal{V}) \to 2^{\mathbb{P}(O \times \mathcal{V})}$.

Definition 12 (Total Specification). *Let $\mathcal{S}$ be a specification. We say that $\mathcal{S}$ is total if*

$$\forall \langle \overline{\mathtt{G}}, \overline{\mathtt{P}} \rangle, \mathtt{op}.\ \exists \mathtt{G}_1, v.\ \overline{\mathtt{G}} = \overline{\mathtt{G}_1} \ \wedge\ \overline{\mathtt{P}} \in \overline{\mathcal{S}(\mathtt{G}_1{}^{\langle \mathtt{op}, v \rangle})}\Big|_{\mathcal{E}_{\mathtt{G}_1}}$$

Intuitively, a specification is total when every projection over O of a context in $\mathbb{C}(O \times \mathcal{V})$, as represented by $\langle \overline{\mathtt{G}}, \overline{\mathtt{P}} \rangle \in \mathbb{C}(O)$, can be extended with the execution of any operation of the data type. This is formalised by stating that for any operation op and any admissible arbitration (sequence of operations) $\overline{\mathtt{P}}$ of a configuration $\overline{\mathtt{G}}$ (once more, labelled only with operations), then $\overline{\mathtt{P}}$ can be extended into an admissible arbitration of the configuration $\overline{\mathtt{G}_1{}^{\langle \mathtt{op}, v \rangle}}$, where $\mathtt{G}_1$ is just one of the possible configurations (the one labelled with the correct return values) whose projection corresponds to $\overline{\mathtt{G}}$.

We remark that a total specification does not prevent the definition of an operation that admits more than one return value in certain configurations, i.e., v in Definition 12 does not need to be unique. For instance, consider the `Generic Register` in Ex. 6, in which operation `rd` may return any of the causally-independent, previously written values. Albeit being total, the specification for `rd` is not deterministic. On the contrary, a specification is deterministic if an operation executed over a configuration admits at most one return value, as formally stated below.

Definition 13 (Deterministic Specification). *Let $\mathcal{S}$ be a specification. We say that $\mathcal{S}$ is deterministic if*

$$\forall \mathtt{G}, \mathtt{op}, v, v'.\ v \neq v' \textit{ implies } \overline{\mathcal{S}(\mathtt{G}^{\langle \mathtt{op}, v \rangle})}\Big|_{\mathcal{E}_{\mathtt{G}}} \cap \overline{\mathcal{S}(\mathtt{G}^{\langle \mathtt{op}, v' \rangle})}\Big|_{\mathcal{E}_{\mathtt{G}}} = \emptyset$$

A weaker notion for determinism could allow the result for an added operation to depend also on the given admissible path. We say that a specification $\mathcal{S}$ is *value-deterministic* if

$$\forall \mathtt{G}, \mathtt{op}, v, v'.\ v \neq v' \wedge\ \mathtt{G} \neq \varepsilon \textit{ implies } \mathcal{S}(\mathtt{G}^{\langle \mathtt{op}, v \rangle})\Big|_{\mathcal{E}_{\mathtt{G}}} \cap \mathcal{S}(\mathtt{G}^{\langle \mathtt{op}, v' \rangle})\Big|_{\mathcal{E}_{\mathtt{G}}} = \emptyset$$

$$\mathcal{S}\left(\begin{array}{c}\langle \texttt{inc}, ok\rangle \\ \downarrow \\ \langle \texttt{rd}, 1\rangle\end{array}\right) = \left\{\begin{array}{c}\langle \texttt{inc}, ok\rangle \\ | \\ \langle \texttt{rd}, 1\rangle\end{array}\right\} \qquad \mathcal{S}\left(\begin{array}{c}\langle \texttt{inc}, fail\rangle \\ \downarrow \\ \langle \texttt{rd}, \bot\rangle\end{array}\right) = \left\{\begin{array}{c}\langle \texttt{inc}, fail\rangle \\ | \\ \langle \texttt{rd}, \bot\rangle\end{array}\right\}$$

(a) (b)

Fig. 9. A value-deterministic and coherent specification.

Finally, we say that a specification is *functional* if it is both deterministic and total.

Example 13. Fig. 9 shows a value-deterministic specification. Although a read operation that follows an increment may return two different values, such difference is explained by the previous computation: in one case the increment succeeds while in the other fails. The specification is however not deterministic because it admits a sequence of operations to be decorated with different return values.

Example 14. It is straightforward to check that the specifications in Ex. 4 and Ex. 5 are deterministic. On the contrary, the specification of the `Generic Register` in Ex. 6 is not even value-deterministic. It suffices to consider a configuration in which a read operation sees two different written values. Similarly, `Set` in Ex. 8 is not deterministic.

The lemma below states a simple criterion for determinism.

Lemma 2. *Let $\mathcal{S}$ be a coherent and deterministic specification. Then*

$$\forall \mathtt{G}_1, \mathtt{G}_2.\ \overline{\mathtt{G}_1} = \overline{\mathtt{G}_2} \text{ implies } \mathtt{G}_1 = \mathtt{G}_2 \vee \overline{\mathcal{S}(\mathtt{G}_1)} \cap \overline{\mathcal{S}(\mathtt{G}_2)} = \emptyset$$

So, if two configurations are annotated with the same operations yet with different values, then their admissible paths are already all different if we disregard return values.

4.2 Correspondence between RDTs and Specifications

This section establishes the connection between RDTs and specifications. We first introduce a mapping from RDTs to specifications.

Definition 14. *Let $\mathcal{F}$ be an* RDT. *We write $\mathbb{S}(\mathcal{F})$ for the specification associated with $\mathcal{F}$, defined as follows*

$$\mathbb{S}(\mathcal{F})(\mathtt{G}) = \textsc{prval}(\mathcal{F}, \mathtt{G})$$

Next result shows that RDTs correspond to specifications that are coherent, functional and saturated.

Lemma 3. *For every* RDT *$\mathcal{F}$, $\mathbb{S}(\mathcal{F})$ is coherent, functional, and saturated.*

The inverse mapping from specifications to RDTs is defined below.

Definition 15. *Let $\mathcal{S}$ be a specification. We write $\mathbb{F}(\mathcal{S})$ for the* RDT *associated with $\mathcal{S}$, defined as follows*

$$\mathbb{F}(\mathcal{S})(\mathtt{op}, \overline{\mathtt{G}}, \overline{\mathtt{P}}) = v \text{ if } \exists \mathtt{G}_1.\ \overline{\mathtt{G}} = \overline{\mathtt{G}_1} \wedge \overline{\mathtt{P}} \in \mathcal{S}(\overline{\mathtt{G}_1{}^{\langle \mathtt{op}, v\rangle}})\Big|_{\mathcal{E}_{\mathtt{G}_1}}$$

Note that $\mathbb{F}(\mathcal{S})$ may not be well-defined for some $\mathcal{S}$, e.g. when $\mathcal{S}$ is not deterministic. The following lemma states the conditions under which $\mathbb{F}(\mathcal{S})$ is well-defined.

Lemma 4. *For every coherent and functional specification $\mathcal{S}$, $\mathbb{F}(\mathcal{S})$ is well-defined.*

The following two results show that RDTs are a particular class of specifications, and hence, provide a fully abstract characterisation of operational RDTs.

Theorem 1. *For every coherent, functional, and saturated specification $\mathcal{S}$, $\mathcal{S} = \mathbb{S}(\mathbb{F}(\mathcal{S}))$.*

Theorem 2. *For every* RDT *$\mathcal{F}$, $\mathcal{F} = \mathbb{F}(\mathbb{S}(\mathcal{F}))$.*

The above characterisation implies that there are data types that cannot be specified as operational RDTs. Consider e.g. `Generic Register` and `Set`, as introduced respectively in Ex. 6 and Ex. 8. As noted in Ex. 14, they are not deterministic. Hence, they cannot be translated as RDTs. We remark that a non-deterministic specification does not imply a non-deterministic conflict resolution, but it allows for underspecification.

5 A categorical account of specifications

In the previous sections we provided a functional characterisation of RDTs. We now proceed on to a denotational account of our formalism by providing a categorical foundation which is amenable to the building of a family of operators on specifications.

5.1 Composing LDAGs

We start by considering a sub-class of morphisms between LDAGs, which account for the evolution of visibility relation by reflecting the information about observed events.

Definition 16 (Past-Reflecting Morphism). *Let $\mathtt{G}_1$ and $\mathtt{G}_2$ be* LDAG*s and $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$ an* LDAG *morphism. We say that $\mathtt{f}$ is* past-reflecting *if*

$$\forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}_1}.\ \mathtt{f}(- \prec \mathtt{e}) = \bigcup_{\mathtt{e}' \in - \prec \mathtt{f}(\mathtt{e})} \mathtt{e}'.$$

We can concisely write $\mathtt{f}(- \prec \mathtt{e}) = - \prec \mathtt{f}(\mathtt{e})$ and spell out the definition as

$$\forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}_1}.\ \forall \mathtt{e}_2 \in \mathtt{G}_2|_{- \prec \mathtt{f}(\mathtt{e})}.\ \exists \mathtt{e}_1 \in \mathtt{G}_1|_{- \prec \mathtt{e}}.\ \mathtt{f}(\mathtt{e}_1) = \mathtt{e}_2$$

It is noteworthy that this requirement boils down to (the dual of) what are called *tp-morphisms* in the literature on algebraic specification theory, which are an instance of *open maps* [9]. As we will see, this property is going to be fundamental in obtaining a categorical characterisation of coherent specifications.

Now, let $\mathcal{G}(\mathcal{L})$ be the category whose objects are LDAGs and arrows are past-reflecting morphisms, and $I(\mathcal{L})$ the sub-category whose arrows are injective morphisms.

Proposition 1 (LDAG **Pullbacks/Pushouts).** *The category $\mathcal{G}(\mathcal{L})$ of* LDAG*s and past-reflecting morphisms has (strict) initial object, pullbacks and pushouts along monos.*

Note that pushout squares along monos are also pullback ones. As often the case, the property concerning pushouts does not hold in $I(\mathcal{L})$, even if a weak form does, since monos are stable under pushouts in $\mathcal{G}(\mathcal{L})$. For the time being, we just remark that these properties guarantee a degree of modularity for our formalism.

We need a last definition before giving a categorical presentation.

Definition 17 (Downward closure). *Let* $\mathtt{G} = \langle \mathcal{E}, \prec, \lambda \rangle$ *be an* LDAG *and* $\mathcal{E}' \subseteq \mathcal{E}$. *We say that* $\mathcal{E}'$ *is downward closed if*

$$\forall \mathtt{e} \in \mathcal{E}'. - \prec \mathtt{e} \subseteq \mathcal{E}'.$$

It is easy to show that for any past-reflecting morphism $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$ the image of $\mathcal{E}_{\mathtt{G}_1}$ along $\mathtt{f}$ is downward closed. Should $\mathtt{f}$ be injective, we strengthen the relationship.

Lemma 5. *An injective morphism* $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$ *is past-reflecting if and only if*

1. $\mathtt{f}(\mathtt{e}_1) \prec_{\mathtt{G}_2} \mathtt{f}(\mathtt{e}_2)$ *implies* $\mathtt{e}_1 \prec_{\mathtt{G}_1} \mathtt{e}_2$*;*
2. $\bigcup_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}_1}} \mathtt{f}(\mathtt{e})$ *is downward closed.*

This result tells us that past-reflecting injective morphisms $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$ are uniquely characterised as such by the properties of the image of $\mathcal{E}_1$ with respect to $\mathtt{G}_2$.

Now, while the initial object of both $\mathcal{G}(\mathcal{L})$ and $I(\mathcal{L})$ is the empty graph ε, the pullback in the latter has an easy characterisation, thanks to the previous lemma. Indeed, let $\mathtt{f}_i : \mathtt{G}_\mathtt{i} \to \mathtt{G}$ be past-preserving injective morphisms, assuming the functions on elements to be identities for the sake of simplicity, and let $\mathcal{E} = \mathcal{E}_{\mathtt{G}_1} \cap \mathcal{E}_{\mathtt{G}_2}$. Then, $\mathtt{G}_1|_{\mathcal{E}} = \mathtt{G}_2|_{\mathcal{E}}$ and they correspond (with the obvious morphisms) to the pullback of $\mathtt{f}_1$ and $\mathtt{f}_2$.

5.2 The model category

We now move to define the model category.

Definition 18 (Morphism Saturation). *Let* $\mathbb{P}(\mathcal{E}_1, \lambda_1)$ *and* $\mathbb{P}(\mathcal{E}_2, \lambda_2)$ *be sets of paths and* $\mathtt{f} : \mathcal{E}_1 \to \mathcal{E}_2$ *an injective function such that* $\lambda_1 = \mathtt{f}; \lambda_2$. *The saturation function* $\mathtt{sat}(-, \mathtt{f})$ *is defined as follows*

$$\mathtt{sat}(\mathtt{P}, \mathtt{f}) = \{\mathtt{Q} \mid \mathtt{Q} \in \mathbb{P}(\mathcal{E}_2, \lambda_2) \text{ and } \mathtt{P} = \mathtt{Q}|_{\mathtt{f}(\mathcal{E}_1)}\}$$

That is, each $\mathtt{Q}$ is the image of $\mathtt{P}$ via a morphism with underlying function $\mathtt{f}$. We can exploit saturation in order to get a simple definition of our model category.

Definition 19 (Path-Set 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 $f : \mathcal{X}_1 \to \mathcal{X}_2$ *is an injective function* $f : \mathcal{E}_1 \to \mathcal{E}_2$ *such that* $\lambda_1 = f; \lambda_2$ *and*

$$\mathcal{X}_2 \subseteq \bigcup_{\mathtt{P} \in \mathcal{X}_1} \mathtt{sat}(\mathtt{P}, \mathtt{f})$$

The property can be stated as

$$\forall \mathtt{P}_2 \in \mathcal{X}_2.\ \exists \mathtt{P}_1 \in \mathcal{X}_1.\ \mathtt{P}_2 \in \mathtt{sat}(\mathtt{P}_1, \mathtt{f})$$

thus each path in $\mathtt{P}_2$ is related to a (unique) path in $\mathtt{P}_1$ via a morphism induced by $\mathtt{f}$. Let $\mathcal{P}(\mathcal{L})$ be the category whose objects are sets of paths over the same elements and labelling (i.e., subsets of $\mathbb{P}(\mathcal{E}, \lambda)$ for some $\mathcal{E}$ and λ), and arrows are path-set morphisms.

Proposition 2 (Path Pullbacks/Pushouts). *The category $\mathcal{P}(\mathcal{L})$ of sets of paths and path-set morphisms has (strict) initial object and pullbacks.*

As for $I(\mathcal{L})$, also $\mathcal{P}(\mathcal{L})$ admits a weak form of pushouts along monos.

Remark 1. The initial object is the set in $2^{\mathbb{P}(\emptyset,\lambda_\emptyset)}$ including only the empty path ε. As for pullbacks, let $\mathtt{f}_i : \mathcal{X}_i \to \mathcal{X}$ be path-set morphisms, assuming the functions on elements to be identities for the sake of simplicity, and let $\mathcal{E} = \mathcal{E}_1 \cap \mathcal{E}_2$. Then, the pullback is the set $\mathcal{X}_1|_{\mathcal{E}} \cup \mathcal{X}_2|_{\mathcal{E}}$ in $2^{\mathbb{P}(\mathcal{E},\lambda)}$ with $\lambda = \lambda_1|_{\mathcal{E}} = \lambda_2|_{\mathcal{E}}$. As for pushouts, let $\mathtt{f} : \mathcal{X} \to \mathcal{X}_i$ be injective path-set morphisms, assuming the functions on elements to be identities for the sake of simplicity, and $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$. Then, the "weak" pushout is the set $\mathcal{X}_1 \otimes \mathcal{X}_2$ in $2^{\mathbb{P}(\mathcal{E},\lambda)}$ with λ the extension of λ_1 and λ_2.

5.3 A categorical correspondence

It is now time to move towards our categorical characterisation of specifications.

First, let us restrict our attention to functors $F : I(\mathcal{L}) \to \mathcal{P}(\mathcal{L})$ that preserve the underlying set of objects, i.e., such that the underlying function on objects Ob_F maps an LDAG $\mathtt{G}$ into a subset of $2^{\mathbb{P}(\mathcal{E}_\mathtt{G},\lambda_\mathtt{G})}$ (and preserves the underling function on path-set morphisms). We also say that F is coherent if $F(\mathtt{G}) = \bigotimes_{\mathtt{e}\in\mathcal{E}_\mathtt{G}} F(\mathtt{G}|_{-\preccurlyeq \mathtt{e}})$ for all LDAGs $\mathtt{G}$. Thus, any such functor F that preserves the initial object (i.e., $F(\varepsilon) = \{\varepsilon\}$) gives raise to a specification: it just suffices to consider the object function $Ob_F : Ob_{I(\mathcal{L})} \to Ob_{\mathcal{P}(\mathcal{L})}$.

Proposition 3. *Let $F : I(\mathcal{L}) \to \mathcal{P}(\mathcal{L})$ be a (coherent) functor preserving the initial object. Then Ob_F is a (coherent) specification.*

For the inverse we need an additional lemma.

Lemma 6. *Let $\mathcal{S}$ be a coherent specification and $\mathcal{E} \subseteq \mathcal{E}_\mathtt{G}$ downward closed. Then $\mathcal{S}(\mathtt{G})|_{\mathcal{E}} \subseteq \mathcal{S}(\mathtt{G}|_{\mathcal{E}})$.*

The lemma above immediately implies the following result.

Proposition 4. *A coherent specification $\mathcal{S}$ induces a coherent functor $\mathbb{M}(\mathcal{S}) : I(\mathcal{L}) \to \mathcal{P}(\mathcal{L})$ preserving the initial object such that $Ob_{\mathbb{M}(\mathcal{S})} = \mathcal{S}$.*

By using Proposition 3 and Proposition 4 we can state the main result of this section.

Theorem 3. *There is a bijection between coherent specifications and coherent functors $I(\mathcal{L}) \to \mathcal{P}(\mathcal{L})$ preserving the initial object.*

6 Conclusions and Future Works

Our contribution proposes a denotational view of replicated data types. While most of the traditional approaches are operational in flavour [4, 7, 8], we strived for a formalism for specifications which could exploit the classical tools of algebraic specification theory. More precisely, we associate to each configuration (i.e., visibility) a set of admissible arbitrations. Differently from those previous approaches, our presentation naturally

accommodates non-deterministic specifications and enables abstract definitions allowing for different strategies in conflict resolution. Our formulation brings into light some properties held by mainstream specification formalisms: beside the obvious property of functionality, they also satisfy coherence and saturation. A coherent specification can neither prescribe an arbitration order between events that are unrelated by visibility nor allow for additional arbitrations over past events when a configuration is extended (i.e., a new top element is added to visibility). Instead, a saturated specification cannot impose any constraint to the arbitration of top elements. Note that saturation does not hold when requiring that admissible arbitrations should be also topological orderings of visibility. Hence, the approaches in [4, 2] generate specifications that are not saturated. We remark that this relation between visibility and arbitration translates in a quite different property in our setting, and this suggests that consistency models defined as relations between visibility and arbitration (e.g., *monotonic* and *causal* consistency) could have alternative characterisations. We plan to explore these connections in future works.

Another question concerns coherence, which prevents a specification from choosing an arbitration order on events that are unrelated by visibility and forbids, e.g., the definition of strategies that arbitrate first the events coming from a particular replica. Consequently, it becomes natural to look for those RDTs and consistency models that are the counterpart of non-coherent specifications, still preserving some suitable notion of causality between events. We do believe that the weaker property $\mathcal{S}(\mathtt{G})|_{-\prec \mathtt{e}} \subseteq \mathcal{S}(\mathtt{G}|_{-\prec \mathtt{e}})$ (that is, no additional arbitration over past events when a configuration is extended) is a worthwhile alternative, accommodating for many examples that impose less restrictions on the set of admissible paths (hence, that may allow more freedom to the arbitration).

These issues might be further clarified by our categorical presentation. Our proposal is inspired by current work on the semantics of nominal calculi [11], and it shares similarities with [10], since our category $\mathcal{G}$ is the sub-category of their $\mathbf{FinSet}^{\rightrightarrows}$ with past-reflecting morphisms. The results on Section 5 focus on a functorial characterisation of specifications. We chose an easy way out for establishing the bijection between functors and specifications by restricting the possible object functions and by defining coherence "on the nose", (i.e., by considering functors F such that $F(\mathtt{G}) \subseteq 2^{\mathbb{P}(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}})}$ and $F(\mathtt{G}) = \bigotimes_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} F(\mathtt{G}|_{-\preccurlyeq \mathtt{e}})$), since requiring the specification to be coherent is needed in order to obtain the functor in Proposition 4. A proper characterisation should depend on the properties of F over the arrows of $\mathcal{G}$ (such as pullback/pushout preservation), instead of the properties of the objects in its image on $\mathcal{P}$.

The same categorical presentation may shed light on suitable operators on specifications. Indeed, this is the usual situation when providing a functorial semantics for a language (see e.g. [1], and the references therein, among many others), and intuitively we have already a freshness operator $F^{\ell}(\mathtt{G}) = F(\mathtt{G}^{\ell})$, along the lines of edge allocation in [10]. We plan to extend these remarks into a full-fledged algebra for specifications.

References

1. Bonchi, F., Buscemi, M.G., Ciancia, V., Gadducci, F.: A presheaf environment for the explicit fusion calculus. Journal of Automated Reasoning 49(2), 161–183 (2012)
2. Burckhardt, S.: Principles of eventual consistency. Foundations and Trends in Programming Languages 1(1-2), 1–150 (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. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2015)
6. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2), 51–59 (Jun 2002)
7. Gotsman, A., Yang, H.: Composite replicated data types. In: Vitek, J. (ed.) ESOP 2015. LNCS, vol. 9032, pp. 585–609. Springer (2015)
8. Gotsman, A., Yang, H., Ferreira, C., Najafzadeh, M., Shapiro, M.: 'cause i'm strong enough: reasoning about consistency choices in distributed systems. In: Bodík, R., Majumdar, R. (eds.) POPL 2016. pp. 371–384. ACM (2016)
9. Joyal, A., Nielson, M., Winskel, G.: Bisimulation and open maps. In: LICS 1993. pp. 418–427. IEEE (1993)
10. Montanari, U., Sammartino, M.: A network-conscious π-calculus and its coalgebraic semantics. Theoretical Computer Science 546, 188–224 (2014)
11. Pitts, A.M.: Nominal Sets: Names and Symmetry in Computer Science. Cambridge University Press (2013)
12. 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)
13. 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)
14. 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)