Science of Computer Programming 167 (2018) 91–113

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

On the semantics and implementation of replicated data types

Fabio Gadducci[a], Hernán Melgratti[b,c], Christian Roldán[b,*]

[a] Università di Pisa, Dipartimento di Informatica, Pisa, Italy
[b] Universidad de Buenos Aires, Facultad de Ciencias Exactas y Naturales, Departamento de Computación, Buenos Aires, Argentina
[c] CONICET-Universidad de Buenos Aires, Instituto de Investigación en Ciencias de la Computación (ICC), Buenos Aires, Argentina

ARTICLE INFO

Article history:
Received 2 November 2017
Received in revised form 7 May 2018
Accepted 20 June 2018
Available online xxxx

Keywords:
Replicated data types
Specification
Implementation correctness

ABSTRACT

Replicated data types (RDTS) concern the specification and implementation of data structures handled by replicated data stores, i.e., distributed data stores that maintain copies of the same data item on multiple devices. A distinctive feature of RDTS is that the behaviour of an operation depends on the state of the replica over which it performs, and hence, its result may differ from replica to replica. Abstractly, RDTS are specified in terms of two relations, *visibility* and *arbitration*. The former establishes whether an operation observes the effects of the execution of another operation, the latter is a total order on operations used to resolve conflicts between operations executed concurrently over different replicas. Traditionally, an operation of an RDT is specified as a function mapping a visibility and an arbitration into the expected result of the operation. This paper recasts such standard approaches into a denotational framework in which a data type is a function mapping visibility into admissible arbitrations. This characterisation provides a more abstract view of RDTS that (i) highlights some implicit assumptions shared in operational approaches to specification; (ii) accommodates underspecification and refinement; (iii) enables a direct characterisation of the correct implementations of an RDT in terms of a simulation relation between the states of a concrete implementation and of the abstract one determined by the specification.

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 [1]. Since many domains cannot renounce availability or avoid 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 is 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`: a memory cell that is read and updated by, respectively, operations `rd` and `wr`. In a replicated scenario, the value obtained when reading a register

[*] Corresponding author.
E-mail addresses: fabio.gadducci@unipi.it (F. Gadducci), hmelgra@dc.uba.ar (H. Melgratti), croldan@dc.uba.ar (C. Roldán).

https://doi.org/10.1016/j.scico.2018.06.003
0167-6423/© 2018 Published by Elsevier B.V.

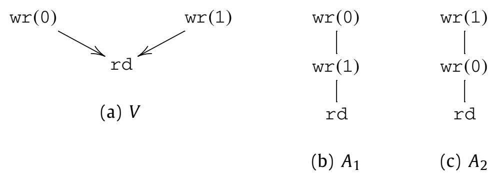

Fig. 1. A scenario for the replicated data type Register.

after two concurrent updates wr(0) and wr(1) (i.e., updates taking place over different replicas) is affected by the way in which updates propagate among replicas: the result might be (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 if a read sees just one update, less so if a read is performed over a replica that knows of both updates, since all replicas should consistently pick the same value among the available ones. A common strategy for registers is that the *last-write wins*: the last update is chosen when several concurrent updates are observed. This strategy implicitly assumes that all the events in a system can be arranged in a total order. Several recent approaches focus on the operational specification of replicated data types [2–9]. Usually, the specification describes the meaning of an operation in terms of two 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 such that $\mathtt{rd}(V, A_1) = \mathtt{1}$ and $\mathtt{rd}(V, A_2) = \mathtt{0}$. We remark that operational approaches require specifications to be *functional*: for every operation, visibility and arbitration, there is 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. In our view, RDTs are functions that map visibility graphs (i.e., configurations) into sets of admissible arbitrations, i.e., all the executions that generate a particular configuration. In this setting, a configuration mapped to an empty set of admissible arbitrations stands for an unreachable configuration, i.e., a configuration that cannot be explained in terms of any arbitration. 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 [2,3], as those specifications that satisfy three properties: besides the evident requirement of being (locally) *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 operation that has not been seen by any other 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 our formulation elegantly accounts for underspecification and refinement, which are standard notions in data type specification.

The notion of implementation correctness, which is central to the theory of abstract data types [10–12], relates the expected behaviour of a family of operations as defined by a specification with the one that is provided by a more concrete realisation. In a replicated scenario, such concrete realisations consist of several replicas that keep their own local state and propagate changes asynchronously. On the one hand, we assume that the behaviour of an implementation is given in terms of two *labelled transition systems* (LTSs): one that describes a single replica and another, which is obtained by composition, that accounts for the joint behaviour of several replicas. Technically, this is achieved by providing a composition operator over LTSs that reflects the adopted communication model. On the other hand, we note that our specifications have an implicit operational interpretation, which describes the expected behaviours of a single replica and of the composition of several replicas. Technically, each specification induces two LTSs: one, called *one-replica*, prescribes the behaviour of a single replica, and another, called *multi-replica*, defines the behaviour of multiple replicas. Then, implementation correctness is defined in terms of simulation relations between the LTSs associated with an implementation, i.e., a replica or a set of replicas, with the LTS corresponding to the specification, i.e., one-replica or multi-replica. We show that implementation correctness is preserved under standard parallel composition (synchronous or asynchronous buffered communication). Consequently, in order to show that an implementation is correct, we only need to show that a single replica is correct. We illustrate the approach with the implementation of a few well-known RDTs.

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 [4]. Section 5 studies the correctness of the replicated data types implementations with respect to our specifications. Finally, in the closing section we draw some conclusions, discuss related works, and highlight further developments.

This paper is a revised and extended version of [13]. We enrich our previous work by providing an approach to assess whether an implementation of an RDT on top of several concurrent replicas is correct (the material in § 5 is completely new to this paper). In addition, we provide full proofs of already published results.

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 $\mathsf{e}, \mathsf{e}', \ldots, \mathsf{e}_1, \ldots$ and $\mathcal{L}$ of labels $\ell, \ell', \ldots, \ell_1, \ldots$