Data in a Concurrent Environment

Egidio Astesiano - Alessandro Giovini - Gianna Reggio
Dept. of Mathematics, University of Genova, Italy

Introduction

Integrating specifications of data types and concurrency seems not only a necessity, but an issue to be handled quite naturally; in L. Lamport's words "... we ignore issues that have already been studied for sequential programs ... A complete specification system would include some method for defining data types, probably using an axiomatic approach." (introduction of his well known paper [L] on the specification of concurrent modules).

However, if we look at recent research on this topics (see, eg, [GV, HJ, B]), we can realize that the integration is very much affected by the basic method for the specification of concurrency. As a result not always the specification of data types is handled so naturally as Lamport's words were suggesting and requiring.

We take the position that processes should in a sense just be a data type as any other one and that an overall approach to specification should permit the specification of a data type as in the case when no concurrency is involved. Thus handling the interaction of processes with data should be part of the specification of the dynamic aspects of a system.

Accordingly to this position, since some years (and in cooperation with other authors) we have elaborated a specification technique by which processes/concurrent systems are special abstract data types, modelling concurrent systems as algebraic transition systems with possibly the specification of a parallel structure (see, eg, [AMRW, AR1, AR2]) (in the following "abstract data type" is abbreviated to adt).

Our technical framework looks naturally apt to accommodate data type specifications. Indeed all the components of a concurrent system are specified as adts. Moreover, since processes are themselves specified as adts, they can be treated as data. As we permit the specification of functions (we use higher-order algebraic specification), we can thus specify data of any complexity and also model systems where processes are exchanged between processes, stored and so on.

After illustrating our specification technique by means of a simple example in section 1, in section 2 we discuss formally the two above aspects. First we show how the embedding of adt specifications into the specification of a concurrent system does not affect their semantics, whatever is the semantics chosen for the system; second we show that this fact also holds when the data are processes/concurrent systems.

Then in section 3 we tackle the main point of the interaction of processes with objects of some data type (this is the issue considered also in [GV, B]). Let us briefly illustrate our technique with the help of a well-known example and by comparing it to two typical and interesting approaches which take a quite different point of view.

This work has been partly carried out with the financial support of CNR-Italy (Progetto Strategico "Software: ricerche di base e applicazioni") and of the ESPRIT project n. 1550 DRAGON.
Assume we want to allow several processes to have access to a queue; whichever formalization we adopt, processes should be able to perform the standard operations on the queue (eg, inserting an element, removing/reading the first element), and moreover the queue should behave as such, ie, it should in some sense implement the axioms of its specification.

Assuming that a framework for specifying and proving properties of processes is available, a possible approach is modelling also a queue as a process, ie, translating a term of sort queue into a process, using the combinators for building processes provided by the framework. This approach has been taken in [GV], and in [HJ], where it is necessary to prove that the process implementing the queue is, in some sense, a queue; this can be done for example by proving that two processes corresponding to provably equal terms of sort queue are semantically equivalent (w.r.t. the chosen semantics of a process).

A similar approach, carried out in [B], consists in modelling a queue as an agent which transforms sequences of input messages into sequences of output messages. So a queue is translated into a function QueueAgent: message* → message* and a function on queue agents is associated with every operation of the queue data type (for example, the operation Add: message × queue → queue becomes a function Add*: message × (message* → message*) → (message* → message*); the definition of these functions is made in term of the corresponding operations on queues, but it has still to be proven that the axioms of the specification of queues hold also on queue agents, what can now be done in purely functional framework.

In our approach we model a queue, even when in parallel with processes, just as a term of sort queue, so that the axioms of the specification automatically hold. A state of a concurrent system is, hence, a collection of processes together with a queue. As a further step, we embody a standard mechanism of defining combinators for handling the queue and we prove that, under suitable assumptions on the context, a process can rely on the fact that the effect of a sequence of operations it performs is the expected one (the one deducible from the axioms). In this way we extend the use of a standard (static) queue in a concurrent environment. By specifying other parts of the concurrent system we can specify typical concurrent features of the queue.

As we will see, we can for example forbid multiple accesses to the queue; alternatively we could enable multiple synchronous readings and also give priority to writing over reading an element. We have hence a nice split of static properties of a queue (specified by means of usual axioms of an adt) and dynamic properties. In section 3 a formal general result is stated, thus justifying the technique we have illustrated (hence we do not need to prove the correctness of the interaction for every data type and system).

A more abstract approach, similar for some principles to our, is in [KP]; but the technique is different, since it uses a fixed language for the specification of processes (axioms in the process algebra style), while our system specification technique is highly parameterized and can accommodate various combinators, levels of abstractions and semantics.

The paper is structured as follows: the reader can either start with the example specification of section 1 or with the discussion of main issues in sections 2 and 3, going back for explanation to section 1 when needed.

Note that our results of sections 2 and 3 are possible because we have a precise formal framework, but the formal definition of this framework is not given here; it can be found, eg, in [AR1, AGRZ]. However the essential theses and technical ideas of this paper are not restricted to a single formal framework.

1 Concurrent System Specification

The purpose of this section is only to present by means of the example specification SIMPLE_CS the basic technical ingredients of our specification method, in order to understand the discussions and results of the following sections. The essential ideas are the following:

• a concurrent system is viewed as a particular kind of transition system, in which a state has an internal structure built starting from the specification of the process components, modelled themselves as transi-