Science of Computer Programming 111 (2015) 190–211

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Time refinement in a functional synchronous language

Louis Mandel [a,*], Cédric Pasteur [b,*,1], Marc Pouzet [d,b,c]

[a] IBM Research, Yorktown Heights, NY, USA
[b] DI, École normale supérieure, Paris, France
[c] INRIA Paris-Rocquencourt, France
[d] Université Pierre et Marie Curie, Paris, France

ARTICLE INFO

Article history:
Received 12 December 2013
Received in revised form 22 June 2015
Accepted 2 July 2015
Available online 10 July 2015

Keywords:
Synchronous languages
Functional languages
Semantics
Type systems

ABSTRACT

Concurrent and reactive systems often exhibit multiple time scales. This situation occurs, for instance, in the discrete simulation of a sensor network where the time scale at which agents communicate is very different from the time scale used to model the internals of an agent.

The paper presents *reactive domains* to simplify the programming of such systems. Reactive domains allow for several time scales to be defined and they enable *time refinement*, that is, the replacement of a system with a more detailed version, without changing its observed behavior.

Our work applies to the ReactiveML language, which extends an ML language with synchronous programming constructs *à la Esterel*. We present an operational semantics for the extended language, a type system that ensures the soundness of programs, and a sequential implementation. We discuss how reactive domains can be used in a parallel implementation.

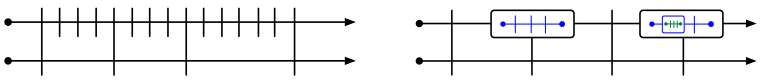
1. Introduction

The concept of logical time greatly simplifies the programming of *concurrent* and *reactive* systems. It is the basis of synchronous languages [1] like Esterel [2]. Its principle is to see the execution of a reactive system as a sequence of reactions, called *instants*, where all communications and computations are considered to be instantaneous during one reaction. This interpretation of time is logical because it does not account for exact computation time and the precise way all the computations are done during a reaction. It has been originally introduced for programming real-time embedded controllers, but it is applicable for a wider range of applications, in particular large scale simulations.

Consider, for example, the simulation of the power consumption in a sensor network [3]. In order to precisely estimate the power consumption, we need to simulate the hardware of certain nodes, in particular the radio. There are now multiple time scales: for example, the time scale of the software (i.e., MAC protocol) is in milliseconds, while the time step of the hardware would be in microseconds. The communication between these time scales must be restricted. E.g., a slow process, whose time scale is in millisecond, cannot observe all the changes of a faster process, whose scale is in microseconds. Said differently, a signal that is produced by a fast process cannot be used to communicate a value with a slower process. Furthermore, depending on the level of precision required for the simulation, it makes sense to be able to replace a precise

* Corresponding authors.
E-mail addresses: lmandel@us.ibm.com (L. Mandel), cedric.pasteur@ansys.com (C. Pasteur), marc.pouzet@ens.fr (M. Pouzet).
[1] Now at ANSYS-Esterel Technologies, Toulouse, France.

http://dx.doi.org/10.1016/j.scico.2015.07.002
0167-6423/



(a) Sampling (b) Reactive domains

Fig. 1. Sampling vs. Reactive domains (each vertical line or box represent one instant of the corresponding clock, horizontal lines represent processes running in parallel).

but costly version of a process that may last several steps by an approximated version, possibly instantaneous. Moreover, this replacement should not impact the way the process interacts with other processes. Such a feature has been called *time* or *temporal refinement* [4].

Synchronous data-flow languages provide a solution to this problem that is based on *sampling*. A slower time scale is obtained by choosing a subset of instants according to a boolean condition. In this paper, we propose *reactive domains*, that allow doing the opposite. Instead of creating a new time scale which is slower than an other one, a reactive domain creates a faster time scale by subdividing an instant of the parent domain. The sequence of instants of a reactive domain stay local to it, that is, they are un-observable from outside, as shown in Fig. 1. Reactive domains make time refinement easy as they allow local computation steps to be hidden (Section 3).

Our work is applied to the ReactiveML language [5], which augments ML with synchronous programming constructs *à la Esterel* (Section 2).[2] We show how to extend the operational semantics of the language to incorporate reactive domains (Section 4). The soundness of programs in the extended setting can be checked using a type-and-effect system, called a *clock calculus*, since it is reminiscent of the one in data-flow synchronous languages [1] (Section 5). Yet, the clock calculus of ReactiveML applies to a language where synchronous constructs are those of Esterel and with ML features. Then, we prove the soundness of the type system with respect to the semantics (Section 6). We also give an overview of the implementation of the extended language and some ideas for parallel execution (Section 7). The article ends with a discussion of several extensions (Section 8) and related work (Section 9).

2. The ReactiveML language

ReactiveML[3] [5,6] is based on the *reactive model* of Boussinot which first appeared in the ReactiveC language [7]. The *reactive model* applies to general purpose languages the principles of the *synchronous model* found in synchronous languages [1].

2.1. Examples

ReactiveML is a reactive extension of ML, so any ML program is also a valid ReactiveML program. The concrete syntax of the language is the one of OCaml,[4] upon which ReactiveML is built. For example, we can define a tree data type and the preorder iteration of a function on a tree by:

```
type 'a tree =
| Empty
| Node of 'a tree * 'a * 'a tree

let rec preorder f t = match t with
| Empty -> ()
| Node(l, v, r) -> f v; preorder f l; preorder f r
```

The type of trees, `'a tree`, is parametrized by the type `'a` of its labels. A tree is either empty, or made of a left child, a label and a right child. The preorder traversal of the tree is implemented with a simple recursive function that applies a given function to the label and recurses first on the left child and then on the right one. We can almost as easily define the level-order traversal of the tree in ReactiveML:

```
let rec process levelorder f t = match t with
| Empty -> ()
| Node (l, v, r) ->
    f v; pause;
    (run levelorder f l || run levelorder f r)
```

This example defines a recursive *process* named `levelorder`. Unlike regular ML expressions, such as a call to `preorder f t`, which is said to be *instantaneous*, the execution of a process can last several logical instants. Here, `levelorder` awaits the next instant by using the `pause` operator and then recursively calls itself on the left and right children in parallel. The `||`

[2] The compiler and the examples mentioned in the paper are available at http://reactiveml.org/scp15.
[3] http://www.reactiveml.org.
[4] http://ocaml.org.