Journal of Logical and Algebraic Methods in Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

"Keep definition, change category" — A practical approach to state-based system calculi

José Nuno Oliveira[a,*], Victor Cacciari Miraldo[b,1]

[a] High Assurance Software Laboratory, INESC TEC and University of Minho, Braga, Portugal
[b] Dep. of Information and Computing Sciences, Universiteit Utrecht, Utrecht, Netherlands

A R T I C L E   I N F O

Article history:
Received 14 October 2014
Received in revised form 24 November 2015
Accepted 26 November 2015
Available online xxxx

Keywords:
Mealy automata
Kleisli category
Relation algebra
Linear algebra
Software calculi

A B S T R A C T

Faced with the need to quantify software (un)reliability in the presence of faults, the semantics of state-based systems is urged to evolve towards quantified (e.g. probabilistic) nondeterminism. When one is approaching such semantics from a categorical perspective, this inevitably calls for some technical elaboration, in a monadic setting.

This paper proposes that such an evolution be undertaken without sacrificing the simplicity of the original (qualitative) definitions, by keeping quantification implicit rather than explicit. The approach is a monad lifting strategy whereby, under some conditions, definitions can be preserved provided the semantics *moves to another category*.

The technique is illustrated by showing how to introduce probabilism in an existing software component calculus, by moving to a suitable category of matrices and using linear algebra in the reasoning.

The paper also addresses the problem of preserving monadic *strength* in the move from original to target (Kleisli) categories, a topic which bears relationship to recent studies in categorial physics.

1. Introduction

The calculus of state-based systems has been a long quest in the theory of computing. The study of deterministic, non-deterministic, probabilistic and weighted automata are steps towards increasingly sophisticated mathematical models intended to express the complexity of the real-life situations they wish to control or mimic.

In the coalgebraic approach [40] a state-based system is regarded as a function of type $S \to \mathbb{F}\,S$ which expresses its *behavior pattern*, that is, how it evolves from the current state (S) to future states ($\mathbb{F}\,S$). In this setting, this evolution is mirrored in the more and more complex *functor* $\mathbb{F}$ which is required to capture the overall behavior:

- $\mathbb{F}\,S = S$ — the system is *deterministic* and total
- $\mathbb{F}\,S = 1 + S$ — the system is *deterministic* but partial (alternative 1 means *failure*)
- $\mathbb{F}\,S = \mathbb{P}\,S$ — the system is *non-deterministic* (where $\mathbb{P}\,S$ is the set of all finite subsets of S)
- $\mathbb{F}\,S = \mathbb{D}\,S$ — the system is *probabilistic* (where $\mathbb{D}\,S$ is the set of all *distributions* of S with finite support).

[*] Corresponding author.
E-mail addresses: jno@di.uminho.pt (J.N. Oliveira), v.cacciarimiraldo@uu.nl (V.C. Miraldo).
URLs: http://haslab.uminho.pt (J.N. Oliveira), http://www.cs.uu.nl (V.C. Miraldo).
[1] Partially supported by grant number UMINHO/BI/9/2014 in the context of FCT Project QAIS (Quantitative Analysis of Information Systems).

http://dx.doi.org/10.1016/j.jlamp.2015.11.007
2352-2208/© 2015 Elsevier Inc. All rights reserved.

The behavior pattern of a complex coalgebraic system may combine two or more functors $\mathbb{F}$ above. A survey by Sokolova [43] identifies no less than thirteen such combinations in the literature, most of them concerned with discrete probabilism. As one would expect, the more $\mathbb{F}$s are involved in the functor expressing the *next state pattern*, the more intricate the corresponding formalization is.

This paper exploits a technique to tame such complexity based on the fact that all functors $\mathbb{F}$ above are *monads* and therefore associated to particular *Kleisli categories* [30]. However, as is well known, not every combination of monads forms a compound monad. Category theory is a generic mathematical framework based on a typed notion of *composition*. The slogan *"keep definition, change category"* emphasizes the stepwise compositionality of the proposed approach: once another monad is combined with $\mathbb{F}$ to capture another aspect of the behavior of the system, one keeps the definition of the previous step and *re-interprets* it inside the Kleisli category of the new monad.

The approach is attractive for two reasons: first, the structure of the "original" definition (written for the simplest case) is preserved and does not get convoluted; second, such Kleisli categories often have good potential for reasoning, as is the case of *relation algebra* (associated to the Kleisli category of the powerset monad) and *linear algebra* (similarly associated to the distribution monad and other weighted semantic models). The cost (if understood as such) is that of expressing the semantics of state-based systems in the *pointfree* language of category theory, putting emphasis on composition and other standard constructs.

Paper structure. This paper is an extension of a previous publication [38] which found its motivation in the need for state-based system calculi to adapt to new trends in circuit design that point towards tolerating some *imperfection* of hardware devices. This calls for semantic models able to cope with faulty behavior in a *quantitative* way, e.g. probabilistic. Sect. 2 and Sect. 3 recall such a motivation. The case study of [38] — the enrichment of a calculus of software components [3] towards fault propagation — is given in Sect. 4 and extended with more results in Sect. 5. The required monad–monad lifting strategy is also enriched in Sect. 8 concerning free monads. More insight into the problem of lifting strong monads is given in Sect. 9. For easy reference, Appendix A gives a minimal set of standard definitions required in the main text. Appendix B gives proofs of auxiliary or lengthy to prove results.

Contribution. This paper proposes that, similarly to what has happened with the increasing role of *relation algebra* in computer science [6,8,42], *linear algebra* be adopted as its natural development where quantitative reasoning is required. Relation algebra and linear algebra share a lot in common once addressed from e.g. a categorial perspective [9,28]. So there is room for evolution rather than radical change.

The contributions of this paper include (a) a case study on such an evolution concerning a calculus of software components [3,4] intended for quantitative analysis of software reliability (sections 4 and 5); (b) a (generic) strategy for reducing the impact of the "probabilistic move" based on re-interpreting state-based system semantics in linear algebra through monadic "Kleisli-lifting", keeping as much of the original semantics definition as possible (sections 6 to 8); (c) a discussion on the loss (across Kleisli-lifting) of the naturality of operations involving pairing, a technical aspect of systems' modeling which needs attention (Sect. 9).

2. Motivation

In the trend towards miniaturization of automated systems the size of circuit transistors cannot be reduced endlessly, as these eventually become unreliable. There is, however, the idea that inexact hardware can be tolerated provided it is "good enough" [26].

Good enough has always been the way engineering works as a broad discipline: why invest in a "perfect" device if a less perfect (and less expensive) alternative suffices? Imperfect circuits will make a certain number of errors, but these will be tolerated if they nevertheless exhibit *almost* the same performance as perfect circuits. This is the principle behind *inexact circuit design* [26], where accuracy of the circuit is exchanged for cost savings (e.g. energy, delay, silicon) in a controlled way.

If unreliable hardware becomes widely accepted on the basis of fault tolerance guarantees, what will the impact of this be on the software layers which run on top of it in virtually any automated system? Running on less reliable hardware, functionally correct (e.g. proven) code becomes faulty and risky. Are we prepared to handle such risk at the software level in the same way it is tackled by hardware specialists? One needs to know how risk propagates across networks of software components so as to mitigate it.

The theory of software design by stepwise refinement already copes with some form of "approximation" in the sense that "vague" specifications are eventually realized by precise algorithms by taking design decisions which lead to (deterministic) code. However, there is a fundamental difference: all input–output pairs of a post-condition in a software specification are *equally* acceptable, giving room for the implementer to choose among them. In the case of imperfect design, one is coping with undesirable, possibly catastrophic outputs which one wishes to prove very unlikely.

In the area of safety critical systems, NASA has defined a *probabilistic risk assessment* (PRA) methodology [44] which characterizes risk in terms of three basic questions: *what can go wrong? how likely is it?* and *what are the consequences?* The PRA process answers these questions by systematically modeling and quantifying those scenarios that can lead to undesired consequences.

Altogether, it seems (as happened with other sciences in the past) that software design needs to become a *quantitative* or *probabilistic* science. Consider concepts such as e.g. *reliability*. From a qualitative perspective, a software system is *reliable*