Theoretical Computer Science 604 (2015) 30–45

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Dijkstra and Hoare monads in monadic computation

Bart Jacobs

Institute for Computing and Information Sciences (iCIS), Radboud University Nijmegen, The Netherlands

ARTICLE INFO

Article history:
Received 3 October 2014
Accepted 12 March 2015
Available online 27 March 2015

Keywords:
Monad
Program semantics
Hoare logic
Weakest precondition

ABSTRACT

The Dijkstra and Hoare monads have been introduced recently for capturing weakest precondition computations and computations with pre- and post-conditions, within the context of program verification, supported by a theorem prover. Here we give a more general description of such monads in a categorical setting. We first elaborate the recently developed view on program semantics in terms of a triangle of computations, state transformers, and predicate transformers. Instantiating this triangle for different computational monads T shows how to define the Dijkstra monad associated with T, via the logic involved.

Subsequently we give abstract definitions of the Dijkstra and Hoare monad, parametrised by a computational monad. These definitions presuppose a suitable (categorical) predicate logic, defined on the Kleisli category of the underlying monad. When all this structure exists, we show that there are maps of monads (Hoare) $\Rightarrow$ (State) $\Rightarrow$ (Dijkstra), all parametrised by a monad T.

1. Introduction

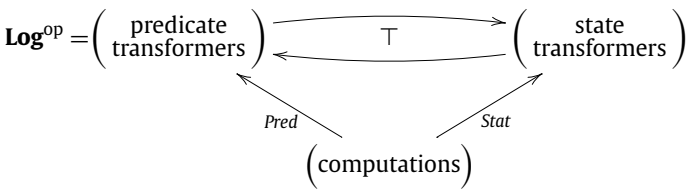
A monad is a categorical concept that is surprisingly useful in the theory of computation. On the one hand it describes a form of computation (such as partial, non-deterministic, or probabilistic), and on the other hand it captures various algebraic structures. Technically, the computations are maps in the Kleisli category of the monad, whereas the algebraic structures are described via the category of so-called Eilenberg–Moore algebras. The Kleisli approach has become common in program semantics and functional programming (notably in the language Haskell), starting with the seminal paper [26]. The algebraic structure captured by the monad exists on these programs (as Kleisli maps), technically because the Kleisli category is enriched over the category of algebras (for suitable monads).

Interestingly, the range of examples of monads has been extended recently from computation to program logic. So-called Hoare monads [27,31] and Dijkstra monads [30] have been defined in a systematic approach to program verification. Via these monads one describes not only a program but also the associated correctness assertions. These monads have been introduced in the language of a theorem prover, but have not been investigated systematically from a categorical perspective. Here we do so not only for the Dijkstra monad (like in [16]), but also for the Hoare monad. We generalise the original definition from [30,27,31] and show that 'Dijkstra' and 'Hoare' monads $\mathfrak{D}_T$ and $\mathfrak{H}_T$ can be associated with various well-known monads T that are used for modelling computations.

Since the Dijkstra and Hoare monads combine both semantics and logic of programs, we need to look at these two areas in a unified manner. From previous work [15] (see also [14]) a view on program semantics and logic emerged involving a triangle of the form:

URL: http://www.cs.ru.nl/B.Jacobs.

http://dx.doi.org/10.1016/j.tcs.2015.03.020
0304-3975/



(1)

The three nodes in this diagram represent categories of which only the morphisms are described. The arrows between these nodes are functors, where the two arrows $\rightleftarrows$ at the top form an adjunction. The two triangles involved should commute. In the case where two up-going 'predicate' and 'state' functors *Pred* and *Stat* in (1) are full and faithful, we have three equivalent ways of describing computations. On morphisms, the predicate functor yields what is called substitution in categorical logic, but what amounts to a weakest precondition operation in program semantics. The upper category on the left is of the form $\mathbf{Log}^{\mathrm{op}}$, where **Log** is some category of logical structures. The opposite category $(-)^{\mathrm{op}}$ is needed because predicate transformers operate in the reverse direction, taking a postcondition to a precondition.

In a setting of quantum computation this translation back-and-forth $\rightleftarrows$ in (1) is associated with the different approaches of Heisenberg (logic-based, working backwards) and Schrödinger (state-based, working forwards), see *e.g.* [9]. In certain cases the adjunction $\rightleftarrows$ forms—or may be restricted to—an equivalence of categories, yielding a duality situation. It shows the importance of duality theory in program semantics and logic; this topic has a long history, going back to [1].

Almost all of our examples of computations are given by maps in a Kleisli category of a monad. In this monadic setting, the right-hand-side of the diagram (1) is the full and faithful 'comparison' functor $\mathcal{K}\ell(T) \to \mathcal{EM}(T)$, for the monad T at hand. This functor embeds the Kleisli category $\mathcal{K}\ell(T)$ in the category $\mathcal{EM}(T)$ of (Eilenberg–Moore) algebras. The left-hand-side takes the form $\mathcal{K}\ell(T) \to \mathbf{Log}^{\mathrm{op}}$, and forms an indexed category (or, if you like, a fibration), and thus a categorical model of predicate logic. The monad T captures computations as maps in its Kleisli category. And via the predicate logic in (1) the 'Dijkstra' and 'Hoare' monads are defined.

One open problem in this area is how to obtain an appropriate categorical logic **Log** for a monad T, yielding a triangle (1). In Sections 6 and 7 we side-step this problem by axiomatising the required categorical logic that is needed for the Dijkstra and Hoare monads. Specifically, we describe the properties that a functor (indexed category) $\mathit{Pred}\colon \mathcal{K}\ell(T) \to \mathbf{Log}^{\mathrm{op}}$ should satisfy so that one can define the Dijkstra and Hoare monad, written as $\mathfrak{D}_T$ and $\mathfrak{H}_T$ respectively, associated with T. It turns out that the Dijkstra monad requires only mild logical properties, whereas the Hoare monad requires much stronger properties. Once they are satisfied we show that there maps of monads:

$$\left(\begin{array}{c}\text{Hoare}\\ \text{monad } \mathfrak{H}_T\end{array}\right) \Longrightarrow \left(\begin{array}{c}\text{state}\\ \text{monad } \mathfrak{S}_T\end{array}\right) \Longrightarrow \left(\begin{array}{c}\text{Dijkstra}\\ \text{monad } \mathfrak{D}_T\end{array}\right)$$

$$\{P\}h\{Q\} \longmapsto h \longmapsto wp(h, -)$$

These maps describe some fundamental relations in the semantics of programs: a Hoare triple $\{P\}h\{Q\}$ is first sent to the program h, as element of the state monad $\mathfrak{S}_T = T(S \times -)^S$ associated with the monad T. In a second step this program h is sent to the weakest precondition operation $wp(h, -)$, mapping a postcondition to a precondition.

We assume that the reader is familiar with the basic concepts of category theory, especially with the theory of monads. The organisation of the paper is as follows: the first three Sections 2–4 elaborate instances of the triangle (1) for non-deterministic, linear & probabilistic, and quantum computation. Subsequently, Section 5 shows how to obtain the Dijkstra monads for the different (concrete) monad examples, and proves in these cases that weakest precondition computation forms a map of monads. This approached is axiomatised in Section 6, starting from a suitable categorical predicate logic. In a similar, but more restricted, axiomatic setting the Hoare monad is defined in Section 7. Finally, Section 8 wraps up with some concluding remarks.

1.1. Notation for monads

We assume the reader is familiar with the notion of monad, see *e.g.* [25,2]. We shall restrict ourselves to monads $T = (T, \eta, \mu)$ on the category **Sets** of sets and functions. We write $\mathcal{K}\ell(T)$ for the Kleisli category of T, with sets X as objects, and 'Kleisli maps' $X \to Y$ given by functions $X \to T(Y)$. We write $J\colon \mathbf{Sets} \to \mathcal{K}\ell(T)$ for the functor given by $J(X) = X$ and $J(f) = \eta \circ f$. Kleisli maps of the form $J(f)$ are often called 'pure'. We use a fat dot $\bullet$ for composition in $\mathcal{K}\ell(T)$, to distinguish it from ordinary composition $\circ$. We recall that $g \bullet f = \mu \circ T(g) \circ f$, and $J(g) \bullet J(f) = J(g \circ f)$.

The Eilenberg–Moore category of the monad T is written as $\mathcal{EM}(T)$. Its objects are 'algebras' $a\colon T(X) \to X$ satisfying $a \circ \eta = \mathrm{id}$ and $a \circ T(a) = a \circ \mu$. There is a canonical functor $\mathit{Stat}\colon \mathcal{K}\ell(T) \to \mathcal{EM}(T)$, sending a set X to the free algebra $\mu\colon T^2(X) \to T(X)$ and a map $f\colon X \to T(Y)$ to the 'Kleisli extension' $f_* = \mu \circ T(f)\colon T(X) \to T(Y)$.

A monad T on **Sets** is automatically strong. There are 'strength' natural transformations $\mathrm{st}_1\colon T(X) \times Y \to T(X \times Y)$ and $\mathrm{st}_2\colon X \times T(Y) \to T(X \times Y)$ defined as:

$$\mathrm{st}_1(u, y) = T(\lambda x.\, \langle x, y\rangle)(u) \qquad \text{and} \qquad \mathrm{st}_2(x, v) = T(\lambda y.\, \langle x, y\rangle)(v).$$