Equational Theories and Monads from Polynomial Cayley Representations

Maciej Piróg(✉), Piotr Polesiuk, and Filip Sieczkowski

University of Wrocław, Wrocław, Poland
mpirog@cs.uni.wroc.pl

Abstract. We generalise Cayley's theorem for monoids by providing an explicit formula for a (multi-sorted) equational theory represented by the type $PX \to X$, where P is an arbitrary polynomial endofunctor with natural coefficients. From the computational perspective, examples of effects given by such theories include backtracking nondeterminism (obtained with the original Cayley representation $X \to X$), finite mutable state (obtained with $n \to X$, for a constant n), and their different combinations (via $n \times X \to X$ or $X^n \to X$). Moreover, we show that monads induced by such theories are implementable using the type formers available in programming languages based on a polymorphic λ-calculus, both as compositions of algebraic datatypes and as continuation-like monads. We give a set-theoretic model of the latter in terms of Barr-dinatural transformations. We also introduce CayMon, a tool that takes a polynomial as an input and generates the corresponding equational theory together with the two implementations of the induced monad in Haskell.

1 Introduction

The relationship between universal algebra and monads has been studied at least since Linton [13] and Eilenberg and Moore [4], while the relationship between monads and the general theory of computational effects (exceptions, mutable state, nondeterminism, and such) has been observed by Moggi [14]. By transitivity, one can study computational effects using concepts from universal algebra, which is the main theme of Plotkin and Power's prolific research programme (see [10,20–24] among many others).

The simplest possible case of this approach is to describe an effect via a finitary equational theory: a finite set of operations (of finite arities), together with a finite set of equations. One such example is the theory of monoids:

$$\begin{array}{ll} \text{Operations:} & \gamma, \quad \varepsilon \\ \text{Equations:} & \gamma(x,\varepsilon) = x, \quad \gamma(\varepsilon,x) = x, \quad \gamma(\gamma(x,y),z) = \gamma(x,\gamma(y,z)) \end{array}$$

M. Bojańczyk and A. Simpson (Eds.): FOSSACS 2019, LNCS 11425, pp. 453–469, 2019.
https://doi.org/10.1007/978-3-030-17127-8_26

The above reads that the signature of the theory consists of two operations: binary γ and nullary ε. The equations state that γ is associative, with ε being its left and right unit.[1] One can also read this theory as a specification of backtracking nondeterminism, in which the order of results matters, where γ is an operation that creates a new computation as a choice between two subcomputations, while ε denotes failure. The connection between the equational theory and the computational effect becomes apparent when we consider the monad of free monoids (that is, the list monad), which is in fact used to form backtracking computations in programming; see, for example, Bird's pearl [1].

This suggests a simple recipe for computational effects: it is enough to come up with an equational theory, and out of the hat comes the induced monad of free algebras that implements the corresponding effect. Such an approach is indeed possible in the category **Set**, where every finitary equational theory admits a free monad, constructed by quotienting terms over the signature by the congruence induced by the equations. However, if we want to implement this monad in a programming language, the situation is not so simple, since in most programming languages (in particular, those without higher inductive types) we cannot generally express this kind of quotients. For instance, to describe a variant of nondeterminism that does not admit duplicate results, we may extend the theory of monoids with an equation stating that γ is idempotent, that is, $\gamma(x, x) = x$. But, unlike in the case of general monoids, the monad induced by the theory of idempotent monoids seems to be no longer directly expressible in, say, Haskell. This means that there is no implementation that satisfies all the equations of the theory "on the nose"—one informal argument is that the representations of $\gamma(x, x)$ and x should be the same whatever the type of x, and this would require a decidable equality test on every type, which is not possible.

Thus, both from the practical viewpoint of programming and as a question on the general nature of equational theories, it makes sense to ask which theories are "simple" enough to induce monads expressible using only the basic type formers, such as (co)products, function spaces, algebraic datatypes, universal and existential quantification. This question seems difficult in general, and to our knowledge there is little work that addresses it. In this paper, we focus on a small piece of this problem: we study a certain subset of such implementable equational theories, and conjure some novel extensions.

The monads that we consider arise from Cayley representations. The overall idea is that if a theory has an expressible, well-behaved (in a sense that we make precise in the paper) Cayley representation, the induced monad also has an expressible implementation. The well-known Cayley theorem for monoids states that every monoid with a carrier X embeds in the monoid of endofunctions $X \to X$. In this paper, we generalise this result: given a polynomial **Set**-endofunctor P with natural coefficients, we provide an explicit formula for an equational theory such that its every algebra with a carrier X embeds in a certain algebra with the carrier given by $PX \to X$. Then, we show that the monad of

[1] Although one usually writes γ as an infix operation, we use a "functional" syntax, since, in the following, the arity of corresponding operations may vary.

free algebras of such a theory can be implemented as a continuation-like monad with the endofunctor given at a set A as:

$$\forall X.(A \to PX \to X) \to PX \to X \tag{1}$$

This type is certainly expressible in programming languages based on polymorphic λ-calculi, such as Haskell.

However, before we can give the details of this construction, we need to address some technical issues. It is easy to notice that there may be more than one "Cayley representation" of a given theory: for example, a monoid X embeds not only in $X \to X$, but also in a "smaller" monoid $X \overset{\gamma}{\rightsquigarrow} X$, by which we mean the monoid of functions of the type $X \to X$ of the shape $a \mapsto \gamma(b, a)$, where $b \in X$. The same monoid X embeds also in a "bigger" monoid $X^2 \to X$, in which we interpret the operations as $\gamma(f, g) = (x, y) \mapsto f(g(x, y), y)$ and $\varepsilon = (x, y) \mapsto x$. What makes $X \to X$ special is that instantiating (1) with $PX = X$ gives a monad that is *isomorphic* to the list monad (note that in this case, the type (1) is simply the Church representation of lists). At the same time, we cannot use $X \overset{\gamma}{\rightsquigarrow} X$ instead of $X \to X$, since (1) quantifies over sets, and thus there is no natural candidate for γ. Moreover, even though we may use the instantiation $PX = X^2$, this choice yields a *different* monad (which we describe in more detail in Sect. 5.4). To sort this out, in Sect. 2, we introduce the notion of *tight Cayley representation*. This notion gives rise to the monad of the following shape, which is a strict generalisation of (1), where R is a **Set**-bifunctor of mixed variance:

$$\forall X.(A \to R(X, X)) \to R(X, X) \tag{2}$$

Formally, all our constructions are set-theoretic—to focus the presentation, the connection with programming languages and type theory is left implicit. Thus, the second issue that we discuss in Sect. 2 is the meaning of the universal quantifier $\forall$ in (1). It is known [27] that polymorphic functions of this shape enjoy a form of dinaturality proposed by Michael Barr (see Paré and Román [16]), called by Mulry *strong* dinaturality [15]. We model the universally quantified types above as collections of Barr-dinatural transformations, and prove that if R is a tight representation, the collection (2) is always a set.

In Sect. 4, we give the formula that defines an equational theory given a polynomial functor P. In general, the theories we construct can be multi-sorted, which is useful for avoiding a combinatory explosion of the induced theories, hence a brief discussion of such theories in Sect. 3. We show that $PX \to X$ is indeed a tight representation of the generated theory. Then, in Sect. 5, we study a number of examples in order to discover what effects are denoted by the generated theories. It turns out that each theory can be seen as a (rather complex, for nontrivial polynomial functors) composition of backtracking nondeterminism and finite mutable state. Moreover, in Sect. 6, we show that the corresponding monads can be implemented not only as continuation-like monads (1), but also in "direct style", using algebraic datatypes.

Since they are parametrised by a polynomial, both the equational theory and its representation consist of many indexed components, so it is not necessarily trivial to get much intuition simply by looking at the formulas. To facilitate this, we have implemented a tool, called CayMon, that generates the theory from a given polynomial, and produces two implementations in Haskell: as a composition of algebraic datatypes and as a continuation-like ("Cayley") monad (1). The tool can be run in a web browser, and is available at http://pl-uwr.bitbucket.io/caymon.

2 Tight Cayley Representations

In this section, we take a more abstract view on the concept of "Cayley representation". In the literature (for example, [2,5,17,25]), authors usually define Cayley representations of different forms of algebraic structures in terms of embeddings. This means that given an object X, there is a homomorphism $\sigma : X \to Y$ to a different object Y, and moreover σ has a retraction (not necessarily a homomorphism) $\rho : Y \to X$ (meaning $\rho \cdot \sigma = \mathsf{id}$). One important fact, which is usually left implicit, is that the construction of Y from X is in some sense functorial. Since we are interested in coming up with representations for many different equational theories, we first identify sufficient properties of such a representation needed to carry out the construction of the monad (2) sketched in the introduction. In particular, we introduce the notion of *tight Cayley representation*, which characterises the functoriality and naturality conditions for the components of the representation.

As for notation, we use $A \to B$ to denote both the type of a morphism in a category, and the set of all functions from A to B (the exponential object in **Set**). Also, for brevity, we write the application of a bifunctor to two arguments, e.g., $G(X, Y)$, without parentheses, as GXY. We begin with the following definition:

Definition 1 (see [16]). *Let $\mathscr{C}, \mathscr{D}$ be categories, and $G, H : \mathscr{C}^{\mathsf{op}} \times \mathscr{C} \to \mathscr{D}$ be functors. Then, a collection of $\mathscr{D}$-morphisms $\theta_X : GXX \to HXX$ indexed by $\mathscr{C}$-objects is called a* Barr-dinatural transformation *if it is the case that for all objects A in $\mathscr{D}$, objects X, Y in $\mathscr{C}$, morphisms $f_1 : A \to GXX$, $f_2 : A \to GYY$ in $\mathscr{D}$, and a morphism $g : X \to Y$ in $\mathscr{C}$,*

if
$$\begin{tikzcd}
 & GXX \arrow[dr, "GXg"] & \\
A \arrow[ur, "f_1"] \arrow[dr, "f_2"'] & & GXY \\
 & GYY \arrow[ur, "GgY"'] &
\end{tikzcd}$$
commutes, then
$$\begin{tikzcd}
 & GXX \arrow[r, "\theta_X"'] & HXX \arrow[dr, "HXg"] & \\
A \arrow[ur, "f_1"] \arrow[dr, "f_2"'] & & & HXY \\
 & GYY \arrow[r, "\theta_Y"] & HYY \arrow[ur, "HgY"'] &
\end{tikzcd}$$
commutes.

An important property of Barr-dinaturality is that the component-wise composition gives a well-behaved notion of vertical composition of two such transformations. The connection between Barr-dinatural transformations and Cayley representations is suggested by the fact, shown by Paré and Román [16], that the collection of such transformations of type $H \to H$ for the **Set**-bifunctor $H(X, Y) = X \to Y$ is isomorphic to the set of natural numbers. The latter, equipped with addition and zero (or the former with composition and the identity transformation, respectively), is simply the free monoid with a single generator, that is, an instance of (1) with $PX = X$ and $A = 1$.

For the remainder of this section, assume that $\mathscr{T}$ is a category, while $F : \mathbf{Set} \to \mathscr{T}$ is a functor with a right adjoint $U : \mathscr{T} \to \mathbf{Set}$. Intuitively, $\mathscr{T}$ is a category of algebras of some theory, and U is the forgetful functor. Then, the monad generated by the theory is given by the composition UF. For a function $f : A \to UX$, we write $\widehat{f} = Uf' : UFA \to UX$, where $f' : FA \to X$ is the contraposition of f via the adjunction (intuitively, the unique homomorphism induced by the freeness of the algebra FA).

Definition 2. *A* tight Cayley representation *of $\mathscr{T}$ with respect to $F \dashv U$ consists of the following components:*

(a) A bifunctor $R : \mathbf{Set}^{\mathsf{op}} \times \mathbf{Set} \to \mathbf{Set}$,
(b) For each set X, an object $\mathbb{R}X$ in $\mathscr{T}$, such that $U\mathbb{R}X = RXX$,
(c) For all sets A, X, Y and functions $f_1 : A \to RXX$, $f_2 : A \to RYY$, $g : X \to Y$, it is the case that

$$\text{if}\quad \begin{tikzcd} & RXX \arrow[dr, "RXg"] & \\ A \arrow[ur, "f_1"] \arrow[dr, "f_2"'] & & RXY \\ & RYY \arrow[ur, "RgY"'] & \end{tikzcd} \quad \text{commutes, then} \quad \begin{tikzcd} & RXX \arrow[dr, "RXg"] & \\ UFA \arrow[ur, "\widehat{f_1}"] \arrow[dr, "\widehat{f_2}"'] & & RXY \\ & RYY \arrow[ur, "RgY"'] & \end{tikzcd} \quad \text{commutes.}$$

(d) For each object M in $\mathscr{T}$, a morphism $\sigma_M : M \to \mathbb{R}(UM)$ in $\mathscr{T}$, such that $U\sigma_M : UM \to R(UM)(UM)$ is Barr-dinatural in M,
(e) A Barr-dinatural transformation $\rho_M : R(UM)(UM) \to UM$, such that $\rho_M \cdot U\sigma_M = \mathsf{id}$,
(f) For each set X, a set of indices I_X and a family of functions $\mathsf{run}_{X,i} : RXX \to X$, where $i \in I_X$, such that $R(RXX)\mathsf{run}_X$ is a jointly monic family, and the following diagram commutes for all X and $i \in I_X$:

$$\begin{tikzcd} RXX \arrow[r, "U\sigma_{\mathbb{R}X}"] \arrow[dr, "R\mathsf{run}_{X,i}X"'] & R(RXX)(RXX) \arrow[d, "R(RXX)\mathsf{run}_{X,i}"] \\ & R(RXX)X \end{tikzcd}$$

Note that the condition (c) states that the objects $\mathbb{R}$ are, in a sense, natural. Intuitively, understanding an object $\mathbb{R}X$ as an algebra, the condition states that the algebraic structure of $\mathbb{R}X$ does not really depend on the set X. The condition (f) may seem rather complicated: the intuition behind the technical formulation is that RXY behaves like a form of a function space (after all, we are interested in abstract *Cayley* representations), and $\mathsf{run}_{X,i}$ is an application to an argument specified by i, as in the example below. In such a case, the joint monicity becomes the extensional equality of functions.

Example 3. Let us check how Cayley representation for monoids fits the definition above: (a) The bifunctor is $RXY = X \to Y$. (b) The $\mathscr{T}$-object for a monoid M is the monoid $M \to M$ with $\gamma(f, g) = f \circ g$ and $\varepsilon = \mathsf{id}$. (c) Given some elements $a, b, \ldots, c \in A$, we need to see that $g \circ f_1(a) \circ f_1(b) \circ \cdots \circ f_1(c) = f_2(a) \circ f_2(b) \circ \cdots \circ f_2(c) \circ g$. Fortunately, the assumption, which in this case becomes $g \circ f_1(a) = f_2(a) \circ g$ for all $a \in A$, allows us to "commute" g from one side of the chain of function compositions to the other. (d) $\sigma_M(a) = b \mapsto \gamma(a, b)$. It is easy to verify that it is a homomorphism. The Barr-dinaturality condition: assuming $f(m) = n$ for some $m \in M$ and $n \in N$, and a homomorphism $f : M \to N$, it is the case that, omitting the U functor, $RfN(\sigma_N(n)) = RfN(\sigma_N(f(m))) = b \mapsto \gamma(f(m), f(b)) = b \mapsto f(\gamma(m, b)) = RMf(\sigma_M(m))$, where the equalities can be explained respectively as: assumption in the definition of Barr-dinaturality, unfolding definitions, homomorphism, unfolding definitions. (e) $\rho_M(f) = f(\varepsilon)$. It is easy to show that it is Barr-dinatural; note that we need to use the fact that $\mathscr{T}$-morphisms (that is, monoid homomorphisms) preserve ε. (f) We define $I_X = X$, while $\mathsf{run}_{X,i}(f) = f(i)$.

The first main result of this paper states that given a tight representation of $\mathscr{T}$ with respect to $F \dashv U$, the monad given by the composition UF can be alternatively defined using a continuation-like monad constructed with sets of Barr-dinatural transformations:

Theorem 4. *For a tight Cayley representation R with respect to $F \dashv U$, elements of the set UFA are in 1-1 correspondence with Barr-dinatural transformations of the type $(A \to RXX) \to RXX$. In particular, this means that the latter form a set. Moreover, this correspondence gives a monad isomorphism between UF and the evident continuation-like structure on* (2)*, given by the unit $(\eta_A(a))_X(f) = f(a)$ and the Kleisli extension $(f^*(k))_X(g) = k_X(a \mapsto (f(a))_X(g))$.*

We denote the set of all Barr-dinatural transformations from the bifunctor $(X, Y) \mapsto A \to RXY$ to R as $\forall X.(A \to RXX) \to RXX$. This gives us a monad similar in shape to the continuation monad, or, more generally, Kock's codensity monad [12] embodied using the formula for right Kan extensions as ends. One important difference with the codensity monad (except, of course, the fact that we have bifunctors on the right-hand sides of arrows) is that we use Barr-dinatural transformations instead of the usual dinatural transformations [3]. Indeed, if we use ends instead of $\forall$, the end $\int_X (A \to RXX) \to RXX$ is given as the collection of all dinatural transformations of the given shape. It is known, however, that even in the simple case when $A = 1$ and $RXY = X \to Y$, this collection is too big to be a set (see the discussion in [16]), hence such end does not exist.

3 Multi-sorted Equational Theories with a Main Sort

The equational theories that we generate in Sect. 4 are multi-sorted, which is useful for trimming down the combinatorial complexity of the result. This turns out to be, in our view, essential in understanding what computational effects they actually represent. In this section, we give a quick overview of what kind of equational theories we work with, and discuss the construction of their free algebras.

We need to discuss the free algebras here, since we want the freeness to be with respect to a forgetful functor to **Set**, rather than to the usual category of sorted sets; compare [26]. This is because we want the equational theories to generate monads on **Set**, as described in the previous section. In particular, we are interested in theories in which one of the sorts is chosen as the *main* one, and work with the functor that forgets not only the structure, but also the carriers of all the other sorts, only preserving the main one. Luckily, this functor can be factored as a composition of two forgetful functors, each with an obvious left adjoint.

In detail, assume a finite set of sorts $S = \{\Omega, K_1, \ldots, K_d\}$ for some $d \in \mathbb{N}$, where Ω is the main sort. The category of sorted sets is simply the category $\mathbf{Set}^{|S|}$, where $|S|$ is the discrete category generated by the set S. More explicitly, the objects of $\mathbf{Set}^{|S|}$ are tuples of sets (one for each sort), while morphisms are tuples of functions. Given an S-sorted finitary theory $\mathfrak{T}$, we denote the category of its algebras as $\mathfrak{T}$-Alg. To see that the forgetful functor from $\mathfrak{T}$-Alg to **Set** has a left adjoint, consider the following composition of adjunctions:

$$\begin{tikzcd}
\mathbf{Set} \arrow[r, bend left, "X \mapsto (X, \emptyset, \ldots, \emptyset)"] & \mathbf{Set}^{|S|} \arrow[l, bend left, "(X, A_1, \ldots, A_d) \mapsto X"] \arrow[r, bend left, "\text{free}"] & \mathfrak{T}\text{-Alg} \arrow[l, bend left, "\text{carriers}"]
\end{tikzcd}$$

This means that the free algebra for each sort has the carrier given by the set of terms of the given sort (with variables appearing only at positions intended for the main sort Ω) quotiented by the congruence induced by the equations. This kind of composition of adjunctions is similar to [18], but in this case the compound right adjoints of the theories given in the next section are monadic.

4 Theories from Polynomial Cayley Representations

In this section, we introduce algebraic theories that are tightly Cayley-represented by $PX \to X$ for a polynomial functor P. Notation-wise, whenever we write $i \leq k$ for a fixed $k \in \mathbb{N}$, we mean that i is a natural number in the range $1, \ldots, k$, and use $[x_i]_{i \leq k}$ to denote a sequence $x_1, \ldots, x_k$. The latter notation is used also in arguments of functions and operations, so $f([x_i]_{i\leq k})$ means $f(x_1, \ldots, x_k)$, while $f(x, [y_i]_{i\leq k})$ means $f(x, y_1, \ldots, y_k)$. We sometimes use double indexing; for example, by $\prod_{i=1}^{k} \prod_{j=1}^{t_i} X_{i,j} \to Y$ for some $[t_i]_{i\leq k}$, we mean the type $X_{1,1} \times \cdots \times X_{1,t_1} \times \cdots \times X_{k,1} \times \cdots \times X_{k,t_k} \to Y$. This is matched by a double-nested notation in arguments, that is, $f([[x_i^j]_{j\leq t_i}]_{i\leq k})$ means $f(x_1^1, \ldots, x_1^{t_1}, \ldots, x_k^1, \ldots, x_k^{t_k})$. Also, whenever we want to repeat an argument k-times, we write $[x]_k$; for example, $f([x]_3)$ means $f(x, x, x)$. Because we use a lot of sub- and superscripts as indices, we do not use the usual notation for exponentiation. This means that x^i always denotes some x at index i, while a k-fold product of some type X, ordinarily denoted X^k, is written as $\prod^k X$. We use the $[\![-]\!]$ brackets to denote the interpretation of sorts and operations in an algebra (that is, a model of the theory). If the algebra is clear from the context, we skip the brackets in the interpretation of operations.

For the rest of the paper, let $d \in \mathbb{N}$ (the number of monomials in the polynomial) and sequences of natural numbers $[c_i]_{i \leq d}$ and $[e_i]_{i \leq d}$ (the coeffcients and exponents respectively) define the following polynomial endofunctor on **Set**:

$$PX = \sum_{i=1}^{d} c_i \times \prod^{e_i} X, \tag{3}$$

where c_i is an overloaded notation for the set $\{1, \ldots, c_i\}$. With this data, we define the following equational theory:

Definition 5. *Assuming d, $[c_i]_{i \leq d}$, and $[e_i]_{i \leq d}$ as above, we define the following equational theory $\mathfrak{T}$:*

– *Sorts:*

$$\begin{aligned} &\Omega && \text{(main sort)} \\ &K_i, \text{ for all } i \leq d && \end{aligned}$$

– *Operations:*

$$\begin{aligned} &\mathsf{cons} : \textstyle\prod_{i=1}^{d} \prod^{c_i} K_i \to \Omega \\ &\pi_i^j : \Omega \to K_i, \text{ for } i \leq d \text{ and } j \leq c_i \\ &\varepsilon_i^j : K_i, \text{ for } i \leq d \text{ and } j \leq e_i \\ &\gamma_i^j : K_j \times \textstyle\prod^{e_j} K_i \to K_i, \text{ for } i, j \leq d \end{aligned}$$

– *Equations:*

$$\begin{aligned} &\pi_i^j(\mathsf{cons}([[x_i^j]_{j \leq c_i}]_{i \leq d})) = x_i^j && \text{(beta-}\pi\text{)} \\ &\mathsf{cons}([[\pi_i^j(x)]_{j \leq c_i}]_{i \leq d}) = x && \text{(eta-}\pi\text{)} \\ &\gamma_i^j(\varepsilon_j^k, [x_t]_{t \leq e_j}) = x_k && \text{(beta-}\varepsilon\text{)} \\ &\gamma_i^i(x, [\varepsilon_i^j]_{j \leq e_i}) = x && \text{(eta-}\varepsilon\text{)} \\ &\gamma_i^j(\gamma_j^k(x, [y_t]_{t \leq e_k}), [z_s]_{s \leq e_j}) = \gamma_i^k(x, [\gamma_i^j(y_t, [z_s]_{s \leq e_j})]_{t \leq e_k}) && \text{(assoc-}\gamma\text{)} \end{aligned}$$

Thus, in the theory $\mathfrak{T}$, there is a main sort Ω, which we think of as corresponding to the entire functor, and one sort K_i for each "monomial" $\prod^{e_i} X$. Then, we can think of Ω as a tuple containing elements of each sort, where each sort K_i has exactly c_i occurrences. The fact that Ω is a tuple, which is witnessed by the cons and π operations equipped with the standard equations for tupling and projections, is not too surprising—one should keep in mind that $\mathfrak{T}$ is a theory represented by the type $PX \to X$, which can be equivalently given as the *product* of function spaces $c_i \times \prod^{e_i} X \to X$ for all $i \leq d$.

Each operation γ_i^j can be used to compose an element of K_j and e_j elements of K_i to obtain an element of K_i. The ε constants can be seen as selectors: in (beta-ε), ε_j^k in the first argument of γ_i^j selects the k-th argument of the sort K_i, while the (eta-ε) equation states that composing a value of K_i with the successive selectors of K_i gives back the original value. The equation (assoc-γ) states that the composition of values is associative in an appropriate sense. In Sect. 5, we provide a reading of the theory $\mathfrak{T}$ as a specification of a computational effect for different choices of d, c_i, and e_i.

Remark 6. If it is the case that $e_i = e_j$ for some $i, j \leq d$, then the sorts K_i and K_j are isomorphic. This means that in every algebra of such a theory, there is an isomorphism of sorts $\varphi : [\![K_i]\!] \to [\![K_j]\!]$, given by $\varphi(x) = \gamma_j^i(x, [\varepsilon_j^k]_{k \leq e_i})$. This suggests an alternative setting, in which instead of having a single $c_i \times \prod^{e_i} X$ comoponent, we can have c_i components of the shape $\prod^{e_i} X$. In such a setting, the equational theory $\mathfrak{T}$ in Definition 5 would be slightly simpler—specifically, there would be no need for double-indexing in the types of cons and π. On the downside, this would obfuscate the connection with computational effects described in Sect. 5 and some conjured extensions in Sect. 7.

The theory $\mathfrak{T}$ has a tight Cayley representation using functions from P, as detailed in the following theorem. This gives us the second main result of this paper: by Theorem 4, the theory $\mathfrak{T}$ is the equational theory of the monad (1). The notation in_i means the i-th inclusion of the coproduct in the functor P.

Theorem 7. *The equational theory $\mathfrak{T}$ from Definition 5 is tightly Cayley-represented by the following data:*

- *The bifunctor $RXY = PX \to Y$,*
- *For a set X, the following algebra:*
  - *Carriers of sorts:*

$$[\![\Omega]\!] = RXX$$
$$[\![K_i]\!] = \prod^{e_i} X \to X$$

  - *Interpretation of operations:*

$$[\![\mathsf{cons}]\!]([[f_k^j]_{j \leq c_k}]_{k \leq d})(\mathsf{in}_i(c, [x_t]_{t \leq e_i})) = f_i^c([x_t]_{t \leq e_i})$$
$$[\![\pi_i^j]\!](f)([x_t]_{t \leq e_i}) = f(\mathsf{in}_i(j, [x_t]_{t \leq e_i}))$$
$$[\![\varepsilon_i^j]\!]([x_t]_{t \leq e_i}) = x_j$$
$$[\![\gamma_i^j]\!](f, [g_k]_{k \leq e_j})([x_t]_{t \leq e_i}) = f([g_k([x_t]_{t \leq e_i})]_{k \leq e_j})$$

- *The homomorphism σ_M for the main sort and sorts K_i:*

$$\sigma_M^{\Omega}(m)(\mathsf{in}_i(c, [x_t]_{t \leq e_i})) = \mathsf{cons}([[\gamma_k^i(\pi_i^c(m), [\pi_k^j(x_t)]_{t \leq e_i})]_{j \leq e_k}]_{k \leq d})$$
$$\sigma_M^{i}(s)([x_t]_{t \leq e_i}) = \mathsf{cons}([[\gamma_k^i(s, [\pi_k^j(x_t)]_{t \leq e_i})]_{j \leq e_k}]_{k \leq d})$$

- *The transformation ρ_M:*

$$\rho_M(f) = \mathsf{cons}([[\pi^j_k(f(\mathsf{in}_k(j, [\mathsf{cons}([w^f_r]_{r<k}, [\varepsilon^t_k]_{c_k}, [w^f_r]_{k<r\leq d})]_{t\leq e_k})))]_{j\leq c_k}]_{k\leq d})$$
$$\text{where } w^f_r = [\pi^c_r(f(\mathsf{in}_r(c, [\varepsilon^j_r]_{j\leq e_r})))]_{c\leq c_r}$$

- *The set of indices $I_X = PX$ and the functions $\mathsf{run}_{X,i}(f) = f(i)$.*

In the representing algebra, it is the case that each $[\![K_i]\!]$ represents one monomial, as mentioned in the description of $\mathfrak{T}$, while $[\![\Omega]\!]$ is the appropriate tuple of representations of monomials, which is encoded as a single function from a coproduct (in our opinion, this encoding turns out to be much more readable on paper), while cons and π are indeed given by tupling and projections. For each $i \leq d$, the function ε^j_i simply returns its j-th argument, while γ is interpreted as the usual composition of multi-argument functions.

Homomorphisms between multi-sorted algebras are defined as operation-preserving functions for each sort, so σ is defined for the sort Ω and for each sort K_i. In general, the point of Cayley representations is to encode an element m of an algebra M using its possible behaviours with other elements of the algebra. It is no different here: for each sort K_i at the c-th occurrence in the tuple, the function σ^Ω packs (using cons) all possible compositions (by means of γ) of values of K_i with the "components" of m (extracted using π). The same happens for each $s \in [\![K_i]\!]$ in $\sigma^i_M(s)$, but there is no need to unpack s, as it is already a value of a single sort.

The transformation ρ_M is a bit more complicated. The argument f is, in general, a function from a coproduct to M, but we cannot simply apply f to one value $\mathsf{in}_i(\ldots)$ for some sort K_i, as we would obviously lose the information about the components in different sorts. This is why we need to apply f to all possible sorts with ε in the right place to ensure that we recover the original value. We extract the information about particular sorts from such values, and combine them using cons. Interestingly, the elements of w^f_r could actually be replaced by any expression of the appropriate sort that is preserved by homomorphisms, assuming that f is also preserved. This is needed to ensure that ρ is Barr-dinatural (the fact that f is preserved by homomorphisms is exactly the assumption in the definition of Barr-dinaturality). For example, if $e_r > 0$ for some $r \leq d$, one can define w^f_r simply as $[\varepsilon^j_r]_{c_r}$ for some $j \leq e_r$. The complicated expression in the definition of w^f_r is a way to produce values also for sorts K_r with $e_r = 0$, which do not have any ε constants.

5 Effects Modeled by Polynomial Representations

Now we describe what kind of computational effects are captured by the theories introduced in the previous section. It turns out that they all are different compositions of finite mutable state and backtracking nondeterminism. These compositions include the two most basic ones: when the state is *local* for each nondeterministic branch, and when it is *global* to the entire computation.

In the following, if there is only one object of a given kind, we skip the indices. For example, if for some i, it is the case that $e_i = 1$, we write ε_i instead of ε_i^1. If $d = 1$, we skip the subscripts altogether.

5.1 Backtracking Nondeterminism via Monoids

We recover the original Cayley theorem for monoids instantiating Theorem 7 with $PX = X$, that is, $d = 1$ and $c_1 = e_1 = 1$. In this case, we obtain two sorts, Ω and K, while the equations (beta-π) and (eta-π) instantiate respectively as follows:

$$\pi(\mathsf{cons}(x)) = x, \quad \mathsf{cons}(\pi(x)) = x$$

This means that both sorts are isomorphic, so one can think of this theory as being single-sorted. Of course, this is always the case if $d = 1$ and $c_1 = 1$. Since $e_1 = 1$, the operation γ is binary and there is a single ε constant. The equations (beta-ε) and (eta-ε) say, respectively, that ε is the left and right unit of γ, that is:

$$\gamma(\varepsilon, x) = x, \quad \gamma(x, \varepsilon) = x$$

Interestingly, the two unit laws for monoids are symmetrical, but in general the (beta-ε) and (eta-ε) equations are not. One should note that the symmetry is already broken when one implements free monoids (that is, lists) in a programming language: in the usual right-nested implementation, the "beta" rule is part of the definition of the `append` function, while the "eta" rule is a theorem. The (assoc-γ) equation instantiates as the associativity of γ:

$$\gamma(\gamma(x, y), z) = \gamma(x, \gamma(y, z))$$

5.2 Finite Mutable State

For $n \in \mathbb{N}$, if we take $PX = n$, that is, $d = 1$, $c_1 = n$ and $e_1 = 0$, we obtain the equational theory of a single mutable cell in which the set of possible states is $\{1, \ldots, n\}$. There are two sorts in the theory: Ω and K. The sort K does not have any interesting structure on its own, as there are no constants ε, and the equation (eta-ε) instantiates to

$$\gamma(x) = x,$$

which means that γ is necessarily an identity. The fact that this theory is indeed the theory of state becomes apparent when we identify Ω as a sort of computations that require some initial state to proceed, and K as computations that produce a final state. Then, the operations $\pi^j : \Omega \to K$ $(j \leq n)$ are the "update" operations, where π^j sets the current state to j, while $\mathsf{cons} : \prod^n K \to \Omega$ is the "lookup" operation, in which the j-th argument is the computation to be executed if the current state is j. The equations (beta-π), for all $j \leq n$, and (eta-π) state respectively:

$$\pi^j(\mathsf{cons}([x_i]_{i\leq n})) = x_j, \quad \mathsf{cons}([\pi^i(x)]_{i\leq n}) = x$$

These equations embody the natural behaviour rules for this limited form of state. The former reads that setting the current state to j and then proceeding with the computation x_i if the current state is i is the same thing as simply proceeding with x_j (note that x_j is of the sort K, hence it does not use the information that the current state has just been updated to j, so there is no need to keep the π^j operation on the right-hand side of the equation). The latter states that if the current state is i and we set the current state to i, it is the same thing as not changing the state at all (note that x does not depend on the current state, as it is the same in every argument of cons).

Interestingly, the presentations of equational theories for state in the literature (for example, [7,23]) are all single-sorted. Such a setting can be recovered by defining the following macro-operations on the sort Ω:

$$\begin{aligned} &\mathsf{put}^j : \Omega \to \Omega && \mathsf{get} : \textstyle\prod^n \Omega \to \Omega \\ &\mathsf{put}^j(x) = \mathsf{cons}([\pi^j(x)]_n) && \mathsf{get}([x_i]_{i\le n}) = \mathsf{cons}([\pi^i(x_i)]_{i\le n}) \end{aligned}$$

The trick here is that the get operation does not change the state (by setting the new state to the current one), while put does not depend on the current state (by having the same computation in every argument of cons). The usual four equations for the interaction of put and get can be obtained by unfolding the definitions and using the (beta-π) and (eta-π) equations:

$$\begin{aligned} &\mathsf{put}^j(\mathsf{put}^k(x)) = \mathsf{put}^k(x) && \mathsf{put}^j(\mathsf{get}([x_i]_{i\le n})) = \mathsf{put}^j(x_j) \\ &\mathsf{get}([\mathsf{get}([x_i]_{i\le n})]_n) = \mathsf{get}([x_i]_{i\le n}) && \mathsf{get}([\mathsf{put}^i(x_i)]_{i\le n}) = \mathsf{get}([x_i]_{i\le n}) \end{aligned}$$

The connection with the implementation of state in programming becomes evident when we take a closer look at the endofunctor of the induced monad from Theorem 4. Consider the following informal calculation:

$$\begin{aligned} &\forall X.(A \to n \to X) \to n \to X \\ \cong\ &\forall X.n \to (A \to n \to X) \to X && \text{(flipping the arguments)} \\ \cong\ &n \to \forall X.(A \to n \to X) \to X && \text{($\forall$ commutes with arrows)} \\ \cong\ &n \to \forall X.(A \times n \to X) \to X && \text{(Curry)} \\ \cong\ &n \to A \times n && \text{(Church)} \end{aligned}$$

This means that not only do we prove that the equational theory corresponds to the usual state monad, but we can actually *derive* the implementation of state as the endofunctor $A \mapsto (n \to A \times n)$.

5.3 Backtracking with Local State

We obtain one way to combine nondeterminism with state using the functor $PX = n \times X$, for $n \in \mathbb{N}$, that is, $d = 1$, $c_1 = n$ and $e_1 = 1$. It has two sorts, Ω and K, which play roles similar to those detailed in the previous section. However, this time K additionally has the structure of a monoid. This gives us the theory of backtracking with *local* state, which means that whenever we make a choice using the γ operation, the computations in each argument carry separate, non-interfering states. In particular, in a computation $\gamma(x, y)$, both subcomputations x and y start with the same state, which is the initial state of the entire computation. This non-interference is guaranteed simply by the system of sorts: the arguments of γ are of the sort K, which means that the stateful computations inside the arguments begin with π, which sets a new state.

We can also obtain a single-sorted theory, similar to the case of the pure state. To the put and get macro-operations, we add choice and failure as follows:

$$\begin{aligned} &\mathsf{choose} : \varOmega \times \varOmega \to \varOmega && \mathsf{fail} : \varOmega \\ &\mathsf{choose}(x, y) = \mathsf{cons}([\gamma(\pi^j(x), \pi^j(y))]_{j \le n}) && \mathsf{fail} = \mathsf{cons}([\varepsilon]_n) \end{aligned}$$

Then, the locality of state can be summarised by the following equality, which is easy to show using the (beta-π) and (eta-π) equations:

$$\mathsf{put}^k(\mathsf{choose}(x, y)) = \mathsf{choose}(\mathsf{put}^k(x), \mathsf{put}^k(y))$$

5.4 Backtracking with Global State

Another way to compose nondeterminism and state is by using *global* state, which is obtained for $n \in \mathbb{N}$ and $PX = X^n$, that is, $d = 1$, $c_1 = 1$, and $e_1 = n$. As in the case of pure backtracking nondeterminism, it means that the sorts $\varOmega$ and K are isomorphic. The intuitive understanding of the expression $\gamma(x, [y_i]_{i \le n})$ is: first perform the computation x, and then the computation y_i, where i is the final state of the computation x. The operation ε^j is: fail, but set the current state to j. In this case, the equations (beta-ε) instantiate to the following for all $j \le n$:

$$\gamma(\varepsilon^j, [y_i]_{i \le n}) = y_j$$

It states that if the first computation fails but sets the state to j, the next step is to try the computation y_j. Note that there is no other way to give a new state than via failure, but this can be circumvented using $\gamma(x, [\varepsilon^k]_n)$ to set the state to k after performing x. The (eta-ε) instantiates to:

$$\gamma(x, [\varepsilon^j]_{j \le n}) = x$$

This reads that if we execute x and then set the current state to the resulting state of x, it is the same as just executing x.

6 Direct-Style Implementation

Free algebras of the theory $\mathfrak{T}$ from Definition 5 can also be presented as terms of a certain shape. They are best described as terms built using the operations from $\mathfrak{T}$ that are well-typed according to the following typing rules, where the types are called Ω, K_i, and P_i for $i \le d$. The type of the entire term is Ω, and $\textsc{Var}(x)$ means that x is a variable.

$$\frac{[[t_i^j : K_i]_{j \le c_i}]_{i \le d}}{\mathsf{cons}([[t_i^j]_{j \le c_i}]_{i \le d}) : \Omega} \qquad \varepsilon_i^j : K_i \qquad \frac{t : P_j \qquad [w_k : K_i]_{k \le e_j}}{\gamma_i^j(t, [w_k]_{k \le e_j}) : K_i} \qquad \frac{\textsc{Var}(x)}{\pi_i^j(x) : P_i}$$

Note that even though variables appear as arguments to the operations π, they are not of the type Ω. This means that the entire term cannot be a variable, as it is always constructed with cons as the outermost operation. Each argument of cons is a term of the type K_i for an appropriate i, which is built out of the operations ε and γ. Note that the first argument of γ is always a variable wrapped in π, while all the other arguments are again terms of the type K_i. Overall, such terms can be captured as the following endofunctors on **Set**, where W^i represents terms of the type K_i, while W^Ω represents terms of the type Ω. By $\mu Y.GY$ we mean the carrier of the initial algebra of an endofunctor G.

$$\begin{aligned} W^i X &= \mu Y. e_i + \textstyle\sum_{j=1}^{d} (\sum^{c_i} X) \times \prod^{e_j} Y \\ W^\Omega X &= \textstyle\prod_{i=1}^{d} \prod^{c_i} W^i X \end{aligned}$$

Clearly, e_i in the definition of W^i represents the ε_i constants, while the second component of the coproduct is a choice between the γ_i operations with appropriate arguments.

It is the case that every term of the sort Ω can be normalised to a term of the type Ω by a term-rewriting system obtained by orienting the "beta" and "assoc" equations left to right, and eta-expanding variables at the top-level:

$$\begin{aligned} &\pi_i^j(\mathsf{cons}([[x_i^j]_{j \le c_i}]_{i \le d})) \rightsquigarrow x_i^j \\ &\gamma_i^j(\varepsilon_j^k, [x_t]_{t \le e_j}) \rightsquigarrow x_k \\ &\gamma_i^j(\gamma_j^k(x, [y_t]_{t \le e_k}), [z_s]_{s \le e_j}) \rightsquigarrow \gamma_i^k(x, [\gamma_i^j(y_t, [z_s]_{s \le e_j})]_{t \le e_k}) \\ &x \rightsquigarrow \mathsf{cons}([[\gamma_i^i(\pi_i^j(x), [\varepsilon_i^k]_{k \le e_i})]_{j \le c_i}]_{i \le d}) \end{aligned}$$

This term rewriting system gives rise to a natural implementation of the monadic structure, where the "beta" and "assoc" rules normalise the two-level term structure, thus implementing the monadic multiplication, while the eta-expansion rule implements the monadic unit.

7 Discussion

The idea for employing Cayley representations to explore implementations of monads induced by equational theories is inspired by Hinze [8], who suggested a connection between codensity monads, Church representation of lists, and the Cayley theorem for monoids. We note that Hinze's discussion is informal, but he suggests using ends, which, as we discuss in Sect. 2, is not sound.

Most of related work follows one of two main paths: it either concentrates on algebraic explanation of monads already used in programming and semantics (for example, [11,19,23]), or on the general connection between different kinds of algebraic theories and computational effects, but without much interest in whether it leads to structures implementable in a programming language. Some exceptions are the construction of the sum of a theory and a free theory [9] or the sum of ideal monads [6]. What we propose in Sect. 4 is a form of a "functional combinatorics": given a type, what kind of algebra describes the possible values?

As our approach veers off the main paths of the recent work on effects, there are many possible directions of future work. One interesting direction would be to generalise **Set**, the base category used throughout this paper, to more abstract categories. After all, we want to talk about structures definable only in terms of (co)products, exponentials, and quantifiers—which are all constructions whose universal properties are singled out and explored using (co)cartesian (or even monoidal) closed categories. However, the current development relies heavily on the particular properties of **Set**, such as extensional equality of functions, which appears in disguise in the condition (f) in Definition 2.

One can also try to extend the type used as a Cayley representation. For example, we could consider the polynomial P in (3) to range over the space of all sets, that is, allow the coefficients c_i to vary over sets rather than natural numbers. In the Cayley representation, it would be enough to consider functions from c_i in place of c_i-fold products. We would immediately gain expressiveness, as the obtained state monad would no longer need to be defined only for a finite set of possible states. On the flip side, this would make the resulting theory infinitary – which, of course, is not uncommon in the field of algebraic treatment of computational effects. However, we decide to stick to the simplest possible setting in this paper, which greatly simplifies the presentation, but still gives us some novel observations, like the fact that the theory of finite state is simply the theory of 2-sorted tuples in Sect. 5.2, or the novel theory of backtracking nondeterminism with global state in Sect. 5.4. Other future extensions that we believe are worth exploring include iterating the construction to obtain a from of a distributive tensor (compare Rivas *et al.*'s [25] "double" representation of near-semirings) or quantifying over more variables, leading to less interaction between sorts.

Acknowledgements. We thank the reviewers for their insightful comments and suggestions.

Maciej Piróg was supported by the National Science Centre, Poland under POLONEZ 3 grant "Algebraic Effects and Continuations" no. 2016/23/P/ST6/02217. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 665778.

Piotr Polesiuk was supported by the National Science Centre, Poland, under grant no. 2014/15/B/ST6/00619.

Filip Sieczkowski was supported by the National Science Centre, Poland, under grant no. 2016/23/D/ST6/01387.

References

1. Bird, R.: Functional pearl: a program to solve Sudoku. J. Funct. Program. **16**(6), 671–679 (2006). http://dx.doi.org/10.1017/S0956796806006058
2. Bloom, S.L., Ésik, Z., Manes, E.G.: A Cayley theorem for Boolean algebras. Am. Math. Monthly **97**(9), 831–833 (1990). http://dx.doi.org/10.2307/2324751
3. Dubuc, E., Street, R.: Dinatural transformations. In: MacLane, S., et al. (eds.) Reports of the Midwest Category Seminar IV, pp. 126–137. Springer, Heidelberg (1970). https://doi.org/10.1007/BFb0060443
4. Eilenberg, S., Moore, J.C.: Adjoint functors and triples. Illinois J. Math. **9**(3), 381–398 (1965). https://projecteuclid.org:443/euclid.ijm/1256068141
5. Ésik, Z.: A Cayley theorem for ternary algebras. Int. J. Algebra Comput. **8**, 311–316 (1998)
6. Ghani, N., Uustalu, T.: Coproducts of ideal monads. ITA **38**(4), 321–342 (2004). https://doi.org/10.1051/ita:2004016
7. Gibbons, J., Hinze, R.: Just do it: simple monadic equational reasoning. In: Chakravarty, M.M.T., Hu, Z., Danvy, O. (eds.) Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, 19–21 September 2011, pp. 2–14. ACM (2011). http://doi.acm.org/10.1145/2034773.2034777
8. Hinze, R.: Kan extensions for program optimisation *Or*: Art and Dan explain an old trick. In: Gibbons, J., Nogueira, P. (eds.) MPC 2012. LNCS, vol. 7342, pp. 324–362. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31113-0_16
9. Hyland, M., Plotkin, G.D., Power, J.: Combining effects: sum and tensor. Theor. Comput. Sci. **357**(1–3), 70–99 (2006). https://doi.org/10.1016/j.tcs.2006.03.013
10. Hyland, M., Power, J.: The category theoretic understanding of universal algebra: Lawvere theories and monads. Electron. Notes Theor. Comput. Sci. **172**, 437–458 (2007). https://doi.org/10.1016/j.entcs.2007.02.019
11. Jaskelioff, M., Moggi, E.: Monad transformers as monoid transformers. Theor. Comput. Sci. **411**(51–52), 4441–4466 (2010). https://doi.org/10.1016/j.tcs.2010.09.011
12. Kock, A.: Continuous Yoneda representation of a small category (1966). Aarhus University preprint. http://home.math.au.dk/kock/CYRSC.pdf
13. Linton, F.: Some aspects of equational categories. In: Eilenberg, S., Harrison, D.K., MacLane, S., Röhrl, H. (eds.) Proceedings of the Conference on Categorical Algebra, pp. 84–94. Springer, Heidelberg (1966). https://doi.org/10.1007/978-3-642-99902-4_3
14. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**(1), 55–92 (1991). https://doi.org/10.1016/0890-5401(91)90052-4
15. Mulry, P.S.: Strong monads, algebras and fixed points. London Mathematical Society Lecture Note Series, pp. 202–216. Cambridge University Press, New York (1992)
16. Paré, R., Román, L.: Dinatural numbers. J. Pure Appl. Algebra **128**(1), 33–92 (1998). http://www.sciencedirect.com/science/article/pii/S0022404997000364
17. Piróg, M.: Eilenberg-Moore monoids and backtracking monad transformers. In: Atkey, R., Krishnaswami, N.R. (eds.) Proceedings of 6th Workshop on Mathematically Structured Functional Programming, MSFP@ETAPS 2016, Eindhoven, Netherlands, 8th April 2016. EPTCS, vol. 207, pp. 23–56 (2016). https://doi.org/10.4204/EPTCS.207.2

18. Piróg, M., Schrijvers, T., Wu, N., Jaskelioff, M.: Syntax and semantics for operations with scopes. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science. LICS 2018, pp. 809–818. ACM, New York (2018). http://doi.acm.org/10.1145/3209108.3209166
19. Piróg, M., Staton, S.: Backtracking with cut via a distributive law and left-zero monoids. J. Funct. Program. **27**, e17 (2017). https://doi.org/10.1017/S0956796817000077
20. Plotkin, G.: Adequacy for algebraic effects with state. In: Fiadeiro, J.L., Harman, N., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 51–51. Springer, Heidelberg (2005). https://doi.org/10.1007/11548133_3
21. Plotkin, G., Power, J.: Adequacy for algebraic effects. In: Honsell, F., Miculan, M. (eds.) FoSSaCS 2001. LNCS, vol. 2030, pp. 1–24. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45315-6_1
22. Plotkin, G.D., Power, J.: Semantics for algebraic operations. Electron. Notes Theor. Comput. Sci. **45**, 332–345 (2001). https://doi.org/10.1016/S1571-0661(04)80970-8
23. Plotkin, G., Power, J.: Notions of computation determine monads. In: Nielsen, M., Engberg, U. (eds.) FoSSaCS 2002. LNCS, vol. 2303, pp. 342–356. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45931-6_24
24. Plotkin, G.D., Power, J.: Computational effects and operations: an overview. Electron. Notes Theor. Comput. Sci. **73**, 149–163 (2004). http://dx.doi.org/10.1016/j.entcs.2004.08.008
25. Rivas, E., Jaskelioff, M., Schrijvers, T.: From monoids to near-semirings: the essence of MonadPlus and alternative. In: Falaschi, M., Albert, E. (eds.) Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming, Siena, Italy, 14–16 July 2015. pp. 196–207. ACM (2015). http://doi.acm.org/10.1145/2790449.2790514
26. Tarlecki, A.: Some nuances of many-sorted universal algebra: a review. Bull. EATCS **104**, 89–111 (2011)
27. Vene, V., Ghani, N., Johann, P., Uustalu, T.: Parametricity and strong dinaturality (2006). https://www.ioc.ee/~tarmo/tday-voore/vene-slides.pdf