Towards a Coalgebraic Chomsky Hierarchy

Sergey Goncharov, Stefan Milius, Alexandra Silva

Towards a Coalgebraic Chomsky Hierarchy
(Extended Abstract)*

Sergey Goncharov[1], Stefan Milius[1], and Alexandra Silva[2]

[1] Lehrstuhl für Theoretische Informatik, Friedrich-Alexander-Universität Erlangen-Nürnberg
[2] Radboud University Nijmegen and Centrum Wiskunde & Informatica, Amsterdam

Abstract. The Chomsky hierarchy plays a prominent role in the foundations of theoretical computer science relating classes of formal languages of primary importance. In this paper we use recent developments on coalgebraic and monad-based semantics to obtain a generic notion of a $\mathbb{T}$-*automaton*, where $\mathbb{T}$ is a monad, which allows the uniform study of various notions of machines (e.g. finite state machines, multi-stack machines, Turing machines, weighted automata). We use the *generalized powerset construction* to define a generic (trace) semantics for $\mathbb{T}$-automata, and we show by numerous examples that it correctly instantiates for some known classes of machines/languages captured by the Chomsky hierarchy. Moreover, our approach provides new generic techniques for studying expressivity power of various machine-based models.

1 Introduction

In recent decades much interest has been drawn to studying generic abstraction devices not only formally generalizing various computation models and tools, but also identifying core principles and reasoning patterns behind them. An example of this kind is given by the notion of *computational monad* [21], which made an impact both on the theory of programming (as an organization tool for denotational semantics [10, 23]) and on the practice (e.g. being implemented as a programming language feature of Haskell [1] and F# [32]). Another example is given by the theory of *coalgebras* [26], which provides a uniform framework for concurrency theory and observational semantics of systems.

In this paper, we use previous work on monads and coalgebras to give a combined (bialgebraic) perspective of the classical automata theory as well as of some less standard models such as weighted automata. This does not only provide a unifying framework to study various computational models but also suggests new perspectives for proving expressivity bounds for various kinds of machines in a generic way.

We base our framework on the notion of a $\mathbb{T}$-*automaton*, i.e. a coalgebra of the form

$$m : X \to B \times (TX)^A,$$

where T is the functor part of a monad $\mathbb{T}$, which we understand as a mathematical abstraction of a *computational effect* (in the sense of [21]) happening in conjunction with

* An extended version of our paper containing all proofs is avaiblable at http://arxiv.org/abs/1401.5277.

state transitions of the automaton; A is the set of inputs; and B is the set of outputs. According to this view, e.g. nondeterminism is the underlying effect of nondeterministic finite state machines. Analogously, we show that certain (nondeterministic) transformations of the pushdown store form the underlying effect of pushdown automata, etc. By instantiating the operational analysis of computational effects from [23] to our setting we arrive at syntactic fixpoint expressions representing $\mathbb{T}$-automata and prove a Kleene-style theorem for them, thus generalizing previous work of the third author [30].

A crucial ingredient of our framework is the *generalized powerset construction* [31], which serves as a coalgebraic counterpart of classical Rabin-Scott determinization algorithm [25]. It allows us to define *trace semantics* of $\mathbb{T}$-automata and fixpoint expressions denoting their behavior.

We give a formal argument indicating that it is unlikely to capture languages beyond $\mathsf{NTIME}(n)$ using coalgebraic (trace) semantics in a straightforward way (i.e., in our case, using the generalized powerset construction) — the phenomenon known before as a property of *real-time machines* [4]. The requirement to be real-time is an inherent coalgebraic phenomenon of reactivity (or productivity), which restricts the class of behaviors that can be modeled. This led us to formulate a more general *observational semantics*, that allows us to take into account internal (or silent τ-)transitions coalgebraically. The latter furthermore enabled us to capture recursively enumerable languages by a special instance of $\mathbb{T}$-automata called tape automata and that are very similar to Turing machines. Capturing any kind of Turing complete formalism by coalgebras has been a long standing open problem, to which the present paper provides an answer. This results brings us closer to having a coalgebraic Chomsky hierarchy and a new abstract understanding of computability theory.

Related work. We build on previous work on coalgebraic modelling and monad-based semantics. Most of the applications of coalgebra to automata and formal languages however addressed rational models (e.g. rational streams, regular languages) from which we note [27] (regular languages and finite automata), [15] (bialgebraic view of Kleene algebra and regular expressions), [30, 20, 22, 3] (coalgebraic regular expressions). More recently, some further generalizations were proposed. In recent work [34] a coalgebraic model of context-free grammars is given, without however an analogous treatment of push-down automata. In [13] some initial results on $\mathbb{T}$-automata over stacks by the first author were presented, which the present work extends considerably.

2 Preliminaries: Deterministic Moore Automata, Coalgebraicaly

In this section we recall the main definitions and existing results on coalgebraic modelling of state machines. This material, as well as the material of the following sections, uses the language of category theory, hence we assume readers to be familiar with basic notions. We use $\mathbf{Set}$ as the main underlying category throughout. Further abstraction from $\mathbf{Set}$ to a more general category, while possible (and often quite straightforward), will not be pursued in this paper. The central notion in this paper is that of an *F-coalgebra* is a pair $(X, f : X \to FX)$ where F is an endofunctor on $\mathbf{Set}$ called *transition type*, X is a set called the *state space* and f is a map called *transition structure*. We shall occasionally identify a coalgebra with its state space if no confusion arises.

Coalgebras of a fixed transition type F form a category whose morphisms are maps of the state spaces commuting with the transition structure: $h\colon X \to Y$ is a coalgebra morphism from $(X, f\colon X \to FX)$ to $(Y, g\colon Y \to FY)$ iff $g \circ h = Fh \circ f$. A final object of this category (if it exists) plays a particularly important role and is called *final coalgebra*. We denote the final F-coalgebra by $(\nu F, \iota\colon \nu F \to F\nu F)$, and write $\hat{f}\colon X \to \nu F$ for the unique homomorphism from (X, f) to $(\nu F, \iota)$.

Our core example is the standard formalization of Moore automata as coalgebras [26]. For the rest of the paper we fix a finite set A of *actions* and a set B of *outputs*. We call the functor $L = B \times (\text{–})^A$ the *language functor* (over A, B). The coalgebras for L are given by a set X of states with a transition structure on X given by maps

$$o : X \to B \qquad \text{and} \qquad \partial_a : X \to X, \qquad (a \in A)$$

where the left-hand map, called the *observation map*, represents an output function (e.g. an acceptance predicate if $B = 2$) and the right-hand maps, called *a-derivatives*, are the next state functions indexed by input actions from A. Finite L-coalgebras are hence precisely classical Moore automata. It is straightforward to extend a-derivatives to w-derivatives with $w \in A^*$ by induction: $\partial_\epsilon(x) = x$; $\partial_{aw}(x) = \partial_a(\partial_w(x))$.

The final L-coalgebra νL always exists and is carried by the set of all *formal power series* B^{A^*}. The transition structure is given by $o(\sigma) = \sigma(\epsilon)$ and $\partial_a(\sigma) = \lambda w.\sigma(aw)$ for every formal power series $\sigma : A^* \to B$. The unique homorphism from an L-coalgebra X to the B^{A^*} assigns to every state $x_0 \in X$ a formal power series that we regard as the *(trace) semantics* of X with x_0 as an initial state. Specifically, if $B = 2$ then finite L-coalgebras are deterministic automata and $B^{A^*} \cong \mathcal{P}(A^*)$ is the set of formal languages on A and the trace semantics assigns to every state of a given finite deterministic automaton the language accepted by that state.

Definition 2.1 (Trace semantics, Trace equivalence). Given an L-coalgebra (X, f) and $x \in X$, we write $[\![-]\!]_X : X \to B^{A^*}$ for the unique L-coalgebra morphism. For every $x \in X$ we call $[\![x]\!]_X$ the *trace semantics* of x (w.r.t. X). *Trace equivalence* identifies exactly those x and y for which $[\![x]\!]_X = [\![y]\!]_Y$ (for possibly distinct coalgebras X and Y); this is denoted by $x \sim y$.

The following result easily follows by definition (see e.g. [27, Theorem 9.1]).

Proposition 2.2. *Given $x \in X$ and $y \in Y$ where X and Y are L-coalgebras, $x \sim y$ iff for any $w \in A^*$, $o(\partial_w(x)) = o(\partial_w(y))$.*

It is well-known that Moore automata, i.e. *finite* L-coalgebras, can be characterized in terms of the formal power series occurring as their trace semantics (see e.g. [27]).

Definition 2.3 (Regular power series). We call a formal power series σ *regular* if the set $\{\partial_w(\sigma) \mid w \in A^*\}$ is finite.

The following result is a rephrasing of a classical result on Moore automata (see e.g. Eilenberg [9]).

Proposition 2.4. *A formal power series is accepted by a Moore automaton if and only if it is regular.*

Remark 2.5. Formal power series are usually considered when $B = k$ is a semiring, in which case one usually also speaks of *rational formal power series* as behaviours of finite weighted automata over k (see e.g. [8]). Our notion of *regular formal power series* (Definition 2.3) generally disagrees with the latter one (unless B is finite) and is in conceptual agreement with such notions as 'regular events' and 'regular trees' [12, 7]. Regular formal power series as the semantics of precisely the finite L-coalgebras are a special instance of a general coalgebraic phenomenon [2, 20]. Let F be any finitary endofunctor on $\mathbf{Set}$. Define the set ϱF to be the union of images of all *finite* F-coalgebras $(X, f : X \to FX)$ under the final morphism $\hat{f} : X \to \nu F$. Then ϱF is a subcoalgebra of νF with an isomorphic transition structure map called the *rational fixpoint* of F. It is (up to isomorphism) uniquely determined by either of the two following universal properties: (1) as an F-coalgebra it is the final locally finite coalgebra and (2) as an F-algebra it is the initial iterative algebra. We refer to [2] for more details.

The characteristic property of regular formal power series can be used as a definitional principle. In fact, given a regular power series σ and assuming that $A = \{a_1, \ldots, a_n\}$, we can view $\{\sigma_1, \ldots, \sigma_k\} = \{\partial_w(\sigma) \mid w \in A^*\}$ as the solution of a system of recursive equations of the form

$$\sigma_i = a_1.\sigma_{i_1} \pitchfork \cdots \pitchfork a_n.\sigma_{i_n} \pitchfork c_i, \quad i = 1, \ldots, k, \tag{2.1}$$

which should be read as follows: for all $1 \leq i, j \leq k$, $\partial_{a_j}(\sigma_i) = \sigma_{i_j}$ and $\sigma_i(\epsilon) = c_i$. Here we introduce $\pitchfork$ as a notation allowing us to syntactically glue together the information about the "head" of a regular formal series and all its derivatives. Reading the $\sigma_1, \ldots, \sigma_k$ as recursion variables, the system (2.1) uniquely determines the corresponding regular power series: for every i it defines $\sigma_i(\epsilon)$ as c_i and for $w = au$ it reduces calculation of $\sigma_i(w)$ to calculation of some $\sigma_j(u)$ — this induction is obviously well-founded.

Any recursive equation (2.1) can be compactly written as

$$\sigma_i = \mu\sigma_i.\ (a_1.\sigma_{i_1} \pitchfork \cdots \pitchfork a_n.\sigma_{i_n} \pitchfork c_i) \tag{2.2}$$

where μ is the fixpoint operator binding the occurrences of σ_i in the right term. One can successively eliminate all the σ_i except σ using the equations (2.2) as assignments and thus obtain a "solution" $\sigma = t$ of (2.1) in σ where t is a closed term given by the following grammar:

$$\gamma ::= \mu X.\ (a_1.\delta \pitchfork \cdots \pitchfork a_n.\delta \pitchfork B) \qquad\qquad \delta ::= X \mid \gamma \tag{2.3}$$

Here X refers to an infinite stock of variables. Equation $\sigma = t$ is then nothing but a condensed representation of system (2.1) and as such it uniquely defines σ. On the other hand, expressions of the form (2.3) suggest a far reaching generalization of classical regular expressions and the fact that they capture exactly regular power series together with Proposition 2.4 can be viewed as a coalgebraic reformulation of Kleene's theorem. This view has been advanced recently (in a rather more general form) in [30, 22] and is of crucial importance for the present work.

Proposition 2.4 in conjunction with the presentation of regular formal power series as expressions (2.3) suggest that every expression gives rise to a finite L-coalgebra

generated by it, whose state space consists of expressions. This is indeed true and can be viewed as a coalgebraic counterpart of the classical Brzozowski's theorem for regular expressions [5]. Given an expression $e = \mu x.\, (a_1.e_1 \pitchfork \cdots a_n.e_n \pitchfork c)$, let

$$o(e) = c \qquad \text{and} \qquad \partial_{a_i}(e) = e_i[e/x]. \tag{2.4}$$

Proposition 2.6. *Let e be a closed expression (2.3). Then the set $\{\partial_w(e) \mid w \in A^*\}$ forms a finite L-coalgebra under the transition structure (2.4).*

3 Monads and Algebraic Theories

In the previous section we have presented a coalgebraic picture of deterministic Moore automata, essentially capturing the Type-3 level of Chomsky hierarchy (modulo the generalization from languages to power series). In order to deal with other levels we introduce *(finitary) monads* and *algebraic theories* as a critical ingredient of our formalization, thus building on top of the recent previous work [17, 31].

In this work we find it easiest to work with monads in the form of *Kleisli triples*.

Definition 3.1 (Kleisli triple). A Kleisli triple $(T, \eta, -^\dagger)$ consists of an object assignment T sending sets to sets, a family of maps $\eta_X : X \to TX$ and an operator, called *Kleisli lifting*, sending any $f : X \to TY$ to $f^\dagger : TX \to TY$. These data are subject to the following axioms: $\eta^\dagger = \mathrm{id}$, $f^\dagger \eta = f$ and $(f^\dagger g)^\dagger = f^\dagger g^\dagger$.

It is well-known that the definition of a monad as a Kleisli triple is equivalent to the usual definition of a monad $\mathbb{T}$ as an endofunctor T equipped with natural transformations $\eta : Id \to T$ (*unit*) and $\mu : T^2 \to T$ (*multiplication*). A $\mathbb{T}$-*algebra* is given by a set X and a map $f : TX \to X$ satisfying standard coherence conditions: $f\eta_X = \mathrm{id}_X$ and $\mu_X Tf = f\mu_X$, and a morphism of $\mathbb{T}$-algebras is just a morphism of algebras for the functor T (see [19]). The category of $\mathbb{T}$-algebras and their morphisms is called *Eilenberg-Moore category of* $\mathbb{T}$ and is denoted by $\mathbf{Set}^{\mathbb{T}}$.

In what follows we occasionally use Haskell-style do-notation: for any $p \in TX$ and $q : X \to TY$ we write $\mathsf{do}\ x \leftarrow p; q(x)$ to denote $q^\dagger(p) \in TY$; and $p \in T(X \times Y)$ we write $\mathsf{do}\, \langle x, y\rangle \leftarrow p; q(x, y)$. This notation allows for a more convenient point-full reasoning with Kleisli morphisms, effectively avoiding potential tedious calculations due to strength. A monad $\mathbb{T}$ is *finitary* if the underlying functor T is finitary, i.e., T preserves filtered colimits. Informally, T being finitary means that T is determined by its action on finite sets. In addition, finitary monads admit an equivalent presentation in terms of (finitary) *algebraic theories*.

Definition 3.2 (Algebraic theory). An *algebraic signature* Σ consists of operation symbols f, each of which comes together with its *arity* n, which is a nonnegative integer — we denote this by $f\colon n \to 1$. Symbols of zero arity are also called *constants*. Terms over Σ are constructed from operations and variables in the usual way. An *algebraic theory* over Σ is given by a set of term equations closed under under inference of the standard equational logic. As usual, an algebraic theory arises as the deductive closure of a set of its *axioms*.

Example 3.3 (Monads, Algebraic theories). Standard examples of computationally relevant monads include (cf. [21]):

– The *finite and unbounded powerset monads* $\mathcal{P}_\omega$ and $\mathcal{P}$. Only the first one is finitary and corresponds to the algebraic theory of join-semilattices with bottom.
– The *store monad* over a store S. The functorial part given as $X \mapsto (X \times S)^S$. Typically, S is the set of maps $L \to V$ from locations L to values V. A function $f : X \to (Y \times S)^S$ represents a computation that takes a value in X and, depending on the current contents of the store S returns a value in Y and a new store content. As shown in [24], if V is finite then the corresponding store monad can be captured by an algebraic theory over operations $\{lookup_l : V \to 1\}_{l \in L}$ and $\{update_{l,v} : 1 \to 1\}_{l \in L, v \in V}$.
– The *continuation monad*. Given any set R, the assignment $X \mapsto (R^X \to R)$ yields a monad under the following definitions: $\eta(x) = \lambda f.\, f(x)$ and $f^\dagger(k) = \lambda c.\, k(\lambda x.\, f(x)(c))$. This monad is known to be non-finitary, unless $R = 1$.

The following class of examples is especially relevant for the coalgebraic modelling of state-based systems.

Example 3.4 (Semimodule monad, Semimodule theory). Given a semiring R, the semimodule monad $\mathbb{T}_R$ assigns to a set X the free left R-semimodule $\langle X \rangle_R$ over X. Explicitly, $\langle X \rangle_R$ consists of all formal linear combinations of the form

$$r_1 \cdot x_1 + \cdots + r_n \cdot x_n \qquad\qquad (r_i \in R, x_i \in X). \tag{3.1}$$

Equivalently, the elements of $\langle X \rangle_R$ are maps $f : X \to R$ with finite support (i.e. with $|\{x \in X \mid f(x) \neq 0\}| < \omega$). The assignment $X \mapsto \langle X \rangle_R$ extends to a monad, which we call the *(free) semimodule monad*: η_X sends any $x \in X$ to $1 \cdot x$ and $\sigma^\dagger(p)$ applies the substitution $\sigma : X \to \langle Y \rangle_R$ to $p \in \langle X \rangle_R$ and renormalizes the result as expected.

The semimodule monad corresponds to the algebraic theory of R-semimodules. Explicitly, we have a constant $\emptyset : 0 \to 1$, a binary operation $+ : 2 \to 1$, and every $r \in R$ gives rise to a unary operation $\bar{r} : 1 \to 1$. Terms of the theory are then build over these operations and modulo the laws of commutative monoids for $+$ and $\emptyset$, plus the following ones of a (left) action of R on a monoid:

$$\begin{aligned} \bar{r}(x + y) &= \bar{r}(x) + \bar{r}(y) & \qquad \bar{r}(\emptyset) &= \emptyset \\ \bar{r}(x) + \bar{s}(x) &= \overline{r + s}(x) & \qquad \bar{0}(x) &= \emptyset \\ \bar{r}(\bar{s}(x)) &= \overline{r \cdot s}(x) & \qquad \bar{1}(x) &= x \end{aligned}$$

It can be shown that any term can by normalized to the form $\bar{r}_1(x_1) + \cdots + \bar{r}_n(x_n)$ and the latter coherently represents the element (3.1) of $\langle X \rangle_R$, which allows us to identify them. Some notable instances of the semimodule monad $\mathbb{T}_R$ for semirings R of interest are the following:

– If R is the Boolean semiring $\{0, 1\}$ then $\mathbb{T}_R$ is (isomorphic to) the finite powerset monad $\mathcal{P}_\omega$.
– If R is the semiring of natural numbers then $\mathbb{T}_R$ is the *multiset* monad: the elements of $\langle X \rangle_R$ are in bijective correspondence with finite multisets over X.

– If R is the interval $[0, +\infty)$ then $\mathbb{T}_R$ is the monad of *finite valuations* used for modelling probabilistic computations [33].

Finally, the following example is critical for modelling the push-down store.

Example 3.5 (Stack monad, Stack theory). Given a finite set of stack symbols Γ, the *stack monad (over Γ)* is the submonad $\mathbb{T}$ of the store monad $(- \times \Gamma^*)^{\Gamma^*}$ for which the elements $\langle r, t\rangle$ of $TX \subseteq (X \times \Gamma^*)^{\Gamma^*}$ satisfy the following restriction: there exists k depending on r, t such that for every $w \in \Gamma^k$ and $u \in \Gamma^*$, $r(wu) = r(w)$ and $t(wu) = t(w)u$. Intuitively, a function $f : X \to TY$ (cf. Example 3.3) has to compute its output in Y and result stack in Γ^* using only a portion of the stack of a predeclared size k that does not depend on the current content of the stack.

The *stack theory* w.r.t. $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ consists of operations $pop : n + 1 \to 1$ and $push_i : 1 \to 1$ $(1 \leq i \leq n)$. The intuition behind these operations is as follows (in each case the variables under an operation represent continuations, i.e. computations that will be performed once the operation has completed its task, cf. [23]):

– $pop(x_1, \ldots, x_n, y)$ proceeds with y if the stack is empty; otherwise it removes the top element of it and proceeds with x_i where $\gamma_i \in \Gamma$ is the removed stack element.
– $push_i(x)$ adds $\gamma_i \in \Gamma$ on top of the stack and proceeds with x.

These operations are subject to the following axioms:

$$\begin{aligned} push_i(pop(x_1, \ldots, x_n, y)) &= x_i \\ pop(push_1(x), \ldots, push_n(x), x) &= x \\ pop(x_1, \ldots, x_n, pop(y_1, \ldots, y_n, z)) &= pop(x_1, \ldots, x_n, z) \end{aligned}$$

As shown in [13] the stack theory is precisely the algebraic theory of the stack monad.

Finally we introduce a monad and the corresponding theory underlying the tape of a Turing machine. We introduce the following notation: given an integer $i \in \mathbb{Z}$, a nonnegative integer k and a map $\sigma : \mathbb{Z} \to \Gamma$, we write $\sigma =_{i\pm k} \sigma'$ ($\sigma =^{i\pm k} \sigma'$) if $\sigma(j) = \sigma'(j)$ for all j such that $|i - j| \leq k$ ($|i - j| > k$).

Definition 3.6 (Tape monad, Tape theory). Let Γ be a finite set of tape symbols. The *tape monad (over Γ)* is the submonad $\mathbb{T}$ of the store monad $(- \times \mathbb{Z} \times \Gamma^{\mathbb{Z}})^{\mathbb{Z}\times\Gamma^{\mathbb{Z}}}$ for which TX consists of exactly those maps $\langle r, z, t\rangle : \mathbb{Z} \times \Gamma^{\mathbb{Z}} \to (X \times \mathbb{Z} \times \Gamma^{\mathbb{Z}})$, which satisfy restriction: there is $k \geq 0$ such that for any $i, j \in \mathbb{Z}$ and $\sigma, \sigma' : \mathbb{Z} \to \Gamma$ if $\sigma =_{i\pm k} \sigma'$ then

$$\begin{gathered} t(i,\sigma) =_{i\pm k} t(i,\sigma'), \qquad r(i,\sigma) = r(i,\sigma'), \qquad |z(i,\sigma) - i| \leq k, \\ t(i,\sigma) =^{i\pm k} \sigma, \qquad z(i,\sigma) = z(i,\sigma'), \qquad t(i,\sigma_{+j}) = t(i+j,\sigma)_{+j}, \\ r(i,\sigma_{+j}) = r(i+j,\sigma), \qquad z(i,\sigma_{+j}) = z(i+j,\sigma) - j. \end{gathered}$$

where σ_{+j} denotes $\sigma \circ (\lambda i.\, i + j)$. The *tape signature* w.r.t. $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ consists of the operations $read : n \to 1$, $write_i : n \to 1$ $(1 \leq i \leq n)$, $lmove : 1 \to 1$, $rmove : 1 \to 1$, which we interpret over any TX as follows:

$$\begin{aligned} [\![read]\!](p_1, \ldots, p_n)(z, \sigma) &= p_{\sigma(z)}(z, \sigma) & [\![lmove]\!](p)(z, \sigma) &= p(z - 1, \sigma) \\ [\![write_i]\!](p)(z, \sigma) &= p(z, \sigma[z \mapsto \gamma_i]) & [\![rmove]\!](p)(z, \sigma) &= p(z + 1, \sigma) \end{aligned}$$

where $\sigma[z \mapsto \gamma]$ overwrites σ with the assignment $z \mapsto \gamma$, that is: $\sigma[z \mapsto \gamma](z) = \gamma$ and $\sigma[z \mapsto \gamma](z') = \sigma(z')$ for $z' \neq z$. The *tape theory* w.r.t. Γ consist of all those equations $p = q$ in the tape signature, which are valid over every TX.

In contrast to the stack theory the tape theory is given indirectly. We leave the question of finding an appropriate complete axiomatization of the tape theory for future work. Meanwhile, we report the following surprising result:

Theorem 3.7. *The tape theory is not finitely axiomatizable.*

Proof (Sketch). It is easy to verify that for any i, j, k the following identity

$$write_i(lmove^k(write_j(rmove^k(x)))) = lmove^k(write_j(rmove^k(write_i(x))))$$

belongs to the tape theory. The left-hand term represents a computation that first writes γ_i at the current head position then moves k steps to the left, writes γ_j and finally moves k steps back to the right; the left-hand computation does the same in a different order. It is then possible for any finite set $\mathcal{A}$ of identities to construct a model that does not satisfy the above identity for a suitable k depending on the size of $\mathcal{A}$. □

4 Reactive $\mathbb{T}$-algebras and $\mathbb{T}$-automata

As in Section 2 we consider a finite set of actions A. We first consider $\mathbb{T}$-algebras which are equipped with a transition structure similar to that of Moore automata but which, in addition, preserves the algebraic structure. Such a transition structure extends a $\mathbb{T}$-algebra with dynamic behaviour (making it into a coalgebra) and hence we call such structures *reactive* $\mathbb{T}$-algebras.

Definition 4.1 (Reactive $\mathbb{T}$-algebra). Let B and X be $\mathbb{T}$-algebras. Then X is a *reactive $\mathbb{T}$-algebra* if X is an L-coalgebra for which $\partial_a : X \to X$ and $o : X \to B$ are $\mathbb{T}$-algebra morphisms.

Observe that the functor $LX = B \times X^A$ lifts to $\mathbf{Set}^{\mathbb{T}}$; in fact, each LX can be equipped with the pointwise $\mathbb{T}$-algebra structure. Thus, reactive $\mathbb{T}$-algebras are simply coalgebras for this lifting of L to $\mathbf{Set}^{\mathbb{T}}$.

Given a $\mathbb{T}$-algebra B, the set of all formal power series B^{A^*} being the final L-coalgebra can be viewed as a reactive $\mathbb{T}$-algebra with the pointwise $\mathbb{T}$-algebra structure, for which ∂_a and o are easily seen to be $\mathbb{T}$-algebra morphisms. Since any reactive $\mathbb{T}$-algebra is an L-coalgebra, reactive $\mathbb{T}$-algebras inherit the general coalgebraic theory from Section 2. In particular, we use for reactive $\mathbb{T}$-algebras the same notions of trace semantics and trace equivalence as for L-coalgebras.

Definition 4.2 ($\mathbb{T}$-automaton). Suppose, B is finitely generated, i.e. there is a finite set B_0 of *generators* and a surjection $TB_0 \to B$ underlying a $\mathbb{T}$-algebra morphism. A $\mathbb{T}$-*automaton* m is given by a triple of maps

$$o^m : X \to B, \qquad t^m : A \times X \to TX, \qquad a^m : TB \to B, \qquad (\bigstar)$$

where a^m is a $\mathbb{T}$-algebra and X is finite. The first two maps in ($\bigstar$) can be aggregated into a coalgebra transition structure, which we write as $m : X \to B \times (TX)^A$ slightly abusing the notation.

A simple nontrivial example of a $\mathbb{T}$-automaton is given by *nondeterministic finite state machines (NFSM)* by taking $B = \{0, 1\}$, $\mathbb{T} = \mathcal{P}_\omega$ and $a^m(s \subseteq \{0, 1\}) = 1$ iff $1 \in s$.

In order to introduce the trace semantics of a $\mathbb{T}$-automaton it suffices to convert it into a reactive $\mathbb{T}$-algebra, for the trace semantics of the latter is settled by Definition 2.1. This conversion is called the *generalized powerset construction* [25] as it generalizes the classical Rabin and Scott NFSM determinization [31]. Observe that LTX is a $\mathbb{T}$-algebra, since TX is the free $\mathbb{T}$-algebra on X and L lifts to $\mathbf{Set}^{\mathbb{T}}$. Therefore, given a $\mathbb{T}$-automaton (★), $m : X \to B \times (TX)^A$ there exists a unique $\mathbb{T}$-algebra morphism $m^\sharp : TX \to B \times (TX)^A$ such that $m^\sharp \eta = m$. This $m^\sharp$ is a reactive $\mathbb{T}$-algebra on TX. Therefore, we define the trace semantics of (★) as follows: $[\![x]\!]_m = [\![\eta(x)]\!]_{TX}$.

Note that the generalized powerset construction does not reduce a $\mathbb{T}$-automaton to a Moore automaton over TX as TX need not be finite, although when it is the case, e.g. $\mathbb{T} = \mathcal{P}_\omega$, the semantics of a $\mathbb{T}$-automaton falls within regular power series, which is precisely the reason why the languages recognized by deterministic and nondeterministic FSM coincide. Surprisingly, all $\mathbb{T}$-automata with a finite B have the same property, which is a corollary of Theorem 7.1 we prove in Section 7.

Proposition 4.3. *For every $\mathbb{T}$-automaton (★) with finite B and $x \in X$, $[\![x]\!]_m : A^* \to B$ is regular.*

We are now ready to introduce fixpoint expressions for $\mathbb{T}$-automata similar to (2.3).

Definition 4.4 (Reactive expressions). Let Σ be an algebraic signature and let B_0 be a finite set. *Reactive expressions* w.r.t. these data are closed terms δ defined according to the following grammar:

$$\begin{aligned} \delta &::= x \mid \gamma \mid f(\delta, \dots, \delta) && (x \in X, f \in \Sigma) \\ \gamma &::= \mu x.\ (a_1.\delta \pitchfork \cdots \pitchfork a_n.\delta \pitchfork \beta) && (x \in X) \\ \beta &::= b \mid f(\beta, \dots, \beta) && (b \in B_0) \end{aligned}$$

where we assume $A = \{a_1, \dots, a_n\}$ and an infinite collection of variables X.

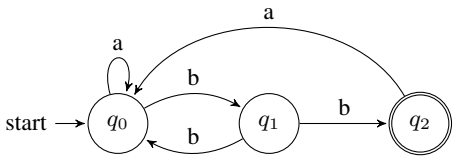
Let $\mathbb{T}$ be a finitary monad, corresponding to an algebraic theory $\mathcal{E}$ over the signature Σ and let B be a finitely generated $\mathbb{T}$-algebra over a finite set of generators B_0. Let us denote by E_{Σ,B_0} the set of all reactive expressions over Σ and B_0. We define a reactive $\mathbb{T}$-coalgebra structure on E_{Σ,B_0}. First, notice that E_{Σ,B_0} is obviously a Σ-algebra. Then we introduce the L-coalgebra structure on E_{Σ,B_0} as follows: first notice that expressions b according to the β-clause in Definition 4.4 are just Σ-terms on the generators from B_0; for every Σ-term t in n variables let $t^B : B^n \to B$ denote the map evaluating t in B and define

$$\begin{aligned} o(f(e_1, \cdots, e_n)) &= f^B(o(e_1), \dots, o(e_n)), \\ \partial_{a_i}(f(e_1, \cdots, e_n)) &= f(\partial_{a_i}(e_1), \dots, \partial_{a_i}(e_n)), \\ o(\mu x.\ (a_1.e_1 \pitchfork \cdots \pitchfork a_n.e_n \pitchfork b)) &= t^B(b_1, \dots, b_k), \\ \partial_{a_i}(\mu x.\ (a_1.e_1 \pitchfork \cdots \pitchfork a_n.e_n \pitchfork b)) &= e_i[\mu x.\ (a_1.e_1 \pitchfork \cdots \pitchfork a_n.e_n \pitchfork b)/x], \end{aligned}$$

where $b = t(b_1, \dots, b_k)$ with $b_1, \dots, b_k \in B_0$.

We call on Definition 2.1 to endow E_{Σ,B_0} with the trace semantics $[\![-]\!] : \mathsf{E}_{\Sigma,B_0} \to B^{A^*}$ and with the trace equivalence relation $\sim$.

$$\begin{cases} q_0 = a.q_0 \pitchfork b.q_1 \pitchfork \bot \\ q_1 = a.\varnothing \pitchfork b.(q_0{+}q_2) \pitchfork \bot \\ q_2 = a.q_0 \pitchfork b.\varnothing \pitchfork \top \end{cases}$$



$$q_0 = \mu x.\,(a.x \pitchfork b.\mu y.\,(a.\varnothing \pitchfork b.(x{+}\mu z.\,(a.x \pitchfork b.\varnothing \pitchfork \top)) \pitchfork \bot) \pitchfork \bot)$$

Fig. 1. A $\mathcal{P}_\omega$-automaton over $A = \{a, b\}$, $B = \{\top, \bot\}$ as a system of recursive definitions (left); as a nondeterministic FSM (right); as a reactive expression (bottom).

Theorem 4.5. *The quotient $\mathsf{E}_{\Sigma,B_0}/{\sim}$ is a reactive $\mathbb{T}$-algebra whose L-coalgebra part is inherited from E_{Σ,B_0} and whose $\mathbb{T}$-algebra part is a quotient of the Σ-algebra structure on E_{Σ,B_0}.*

The following theorem is the main result of this section — a Kleene type theorem. Like its classical counterpart it allows to convert $\mathbb{T}$-automata to closed expressions and vice versa.

Theorem 4.6 (Kleene theorem). *For any expression $e \in \mathsf{E}_{\Sigma,B_0}$ there is a corresponding $\mathbb{T}$-automaton (★) and $x \in X$ such that $[\![e]\!] = [\![x]\!]_m$; and conversely for any $\mathbb{T}$-automaton (★) and $x \in X$ there is an expression $e \in \mathsf{E}_{\Sigma,B_0}$ such that $[\![e]\!] = [\![x]\!]_m$.*

Fig. 1 depicts a simple instance of the general correspondence established by Theorem 4.6 in the particular standard case of nondeterministic FSM.

5 $\mathbb{T}$-automata: Examples

As indicated above, a nondeterministic FSM is a specific case of a $\mathbb{T}$-automaton under $B = 2$ and $\mathbb{T} = \mathcal{P}_\omega$. More generally, we have the following definition.

Definition 5.1 (Weighted $\mathbb{T}$-automata). *Weighted $\mathbb{T}$-automaton* is a $\mathbb{T}$-automaton (★) with $\mathbb{T}$ being the semimodule monad for the semiring R (see Example 3.4).

Let R be the underlying semiring of a semimodule monad $\mathbb{T}$. Besides $R = B = 2$ in which case we obtain nondeterministic FSMs, we obtain standard weighted automata [8] under $R = B = N$ (B is the free $\mathbb{T}$-algebra finitely generated by $\{1\}$).

Weighted $\mathbb{T}$-automata can be further generalized as follows. We call a monad *additive* (cf. [6]) if the corresponding algebraic theory supports operations $+ : 2 \to 1$ and $\emptyset : 0 \to 1$ subject to the axioms of commutative monoids. We call a $\mathbb{T}$-automaton *additive* if $\mathbb{T}$ is additive. Additive automata allow for a more relaxed syntax of reactive expressions. Specifically, we define *additive reactive expressions* as closed *guarded* expressions over an additive theory given by the grammar

$$\gamma ::= b \mid x \mid \mu x.\,\gamma \mid a.\gamma \mid f(\gamma, \ldots, \gamma), \tag{5.1}$$

where guardedness means that for any subterm of the form $\mu x.\, e$ (the recursive call of) x is guarded in e, which is defined by induction over e as follows: x is guarded in b, in any variable $x' \neq x$, in any $\mu x.\, e'$ and in any $a.e'$; and x is guarded in $f(e_1, \ldots, e_n)$ whenever x is guarded in each of the e_i.

Given a reactive expression we obtain an additive reactive expression by replacing recursively each $\pitchfork$ with $+$. Conversely, any additive reactive expression can be transformed to a reactive one. The latter transformation is inverse to the former modulo $\sim$. We now give one example of an additive $\mathbb{T}$-automaton, which is not a weighted $\mathbb{T}$-automaton.

Example 5.2 (Segala $\mathbb{T}$-automata). *(Simple) Segala systems* [28, 29] are systems combining probability and nondeterminism and are essentially coalgebras of transition type $\mathcal{P}(\mathcal{D} \times A) \cong (\mathcal{P}\mathcal{D})^A$ where $\mathcal{D}$ is a probability distribution functor. Although $\mathcal{P}\mathcal{D}$ is not a monad, as elaborated in [16], it can be modelled by a monad $\mathbb{T}$ whose functorial part is the composition CM of two functors given as follows: for any X, MX are finite valuations over X (see Example 3.4); for any semimodule U, $C(U)$ consists of all subsets of U, which are *convex* and nonempty. Convexity of a set S here means that a convex combination $p_1 \cdot \xi_1 + \cdots + p_n \cdot \xi_n$, i.e. $\sum_i p_i = 1$, belongs to S once $\xi_i \in S$ for any i. *Segala $\mathbb{T}$-automata* generalize non-deterministic automata by replacing the powerset functor $\mathcal{P}$ with CM. Concretely, in the generic definition (★) we take $B = 2$ and $\mathbb{T}$ defined as above.

A radically different kind of examples is offered by submonads of the store monad. A prominent instance of such is the stack monad (Example 3.5), which we use for modelling push-down automata.

Definition 5.3 (Stack $\mathbb{T}$-automaton). *Stack $\mathbb{T}$-automaton* is a $\mathbb{T}$-automaton (★) for which $\mathbb{T}$ is the stack monad over Γ; B is the set of predicates over Γ^* consisting of all those $p \in 2^{\Gamma^*}$ for each of which there is k such that $p(wu) = p(w)$ whenever $|w| \geq k$; $a^{\mathcal{m}} : TB \to B$ is given by evaluation; it restricts the morphism $\mathsf{ev}^{\Gamma^*} : (2^{\Gamma^*} \times \Gamma^*)^{\Gamma^*} \to 2^{\Gamma^*}$, where $\mathsf{ev} : 2^{\Gamma^*} \times \Gamma^* \to 2$ is the evaluation morphism: $a^{\mathcal{m}}(r, t)(s) = r(s)(t(s))$.

Intuitively, $o^{\mathcal{m}} : X \to B \subseteq 2^{\Gamma^*}$ models the acceptance condition by final states and the stack content. As B obeys essentially the same constraints as TX, scanning an unbounded portion of the stack by $o^{\mathcal{m}}$ is disallowed.

Theorem 5.4. *Let $\mathcal{m}$ be a stack $\mathbb{T}$-automaton. Then for any $x_0 \in X$ and any $\gamma_0 \in \Gamma$, $\{w \in A^* \mid [\![x_0]\!]_{\mathcal{m}}(w)(\gamma_0)\}$ is a real-time deterministic context-free language. The converse is also true: for any real-time deterministic context-free language $\mathcal{L} \subseteq A^*$ there exists a stack $\mathbb{T}$-automaton (★) such that $\mathcal{L}$ can be represented as the above formal language with some $x_0 \in X$, $\gamma_0 \in \Gamma$.*

As we shall see later, it is not difficult to obtain an analogous characterization of context-free languages for which the "real-time" clause is dropped (essentially because for push-down automata the restriction of being real-time is vacuous). However, as we shall see in the next section (Theorem 6.5), this restriction, being somewhat inherent for coalgebraic models, presents an actual boundary for capturing by $\mathbb{T}$-automata formal languages beyond the context-free ones.

6 Monad Tensors for Store and Nondeterminism

Tensor products of monads (resp. algebraic theories) have been introduced by Freyd [11] in the context of universal algebra. Later, computational relevance of this operation has been demonstrated by Hyland et al. [14]. Here, we use tensors of monads as a tool for studying $\mathbb{T}$-automata.

Definition 6.1 (Tensor). Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two algebraic theories. Then the tensor product $\mathcal{E} = \mathcal{E}_1 \otimes \mathcal{E}_2$ is the algebraic theory, whose equations are obtained by joining the equations of $\mathcal{E}_1$ and $\mathcal{E}_2$ and adding for any $f : n \to 1$ of $\mathcal{E}_1$ and any $g : m \to 1$ of $\mathcal{E}_2$ the following axiom

$$f(g(x_1^1, \ldots, x_m^1), \ldots, g(x_1^n, \ldots, x_m^n)) = g(f(x_1^1, \ldots, x_1^n), \ldots, f(x_m^1, \ldots, x_m^n))$$

called the *tensor laws*. Given two finitary monads $\mathbb{T}_1$ and $\mathbb{T}_2$, their tensor product arises from the algebraic theory $\mathcal{E}_{\mathbb{T}_1} \otimes \mathcal{E}_{\mathbb{T}_2}$.

Intuitively, tensor product of two monads captures a noninterfering combination of the corresponding computational effects. In the present work we shall use two kinds of tensor products: (1) tensors with *submonads of the store monad* and (2) tensors with *semimodule monads*.

It has been shown in [14] that tensoring with the store monad is equivalent to the application of the store monad transformer sending any monad $\mathbb{T}$ to the *store monad transform* $\mathbb{T}_S$ whose functorial part is given by $T_S X = T(X \times S)^S$. Here we establish a similar result about the stack monad (Example 3.5).

Proposition 6.2. *Let $\mathbb{S}$ be the stack monad over Γ. Then for any finitary $\mathbb{T}$, $\mathbb{S} \otimes \mathbb{T}$ is the submonad $\mathbb{R}$ of the store monad transform of $\mathbb{T}$ with Γ^* as the store, for which $p : \Gamma^* \to T(X \times \Gamma^*)$ is in $\mathbb{R}X$ iff there exists m such that $p(su) = \mathsf{do}\, \langle x, s' \rangle \leftarrow p(s); \eta\langle x, s'u \rangle$ whenever $|s| \geq m$; and the monad structure is inherited from the monad transform.*

One can thus combine two stacks by computing the tensor square of the stack monad. Specifically, the resulting monad $\mathbb{T}$ has maps $\langle r, t_1, t_2 \rangle : \Gamma^* \times \Gamma^* \to X \times \Gamma^* \times \Gamma^*$ as inhabitants of TX. This allows one to define $\mathbb{T}$-stack automata over two and more stacks analogously to the one-stack case from Definition 5.3. Before we do this formally in Definition 6.4 we discuss the perspectives of forming tensors with semimodule monads.

Proposition 6.3 (Freyd [11]). *Tensor product of any finitary monad with a semimodule monad is isomorphic to some semimodule monad.*

Proposition 6.3 in conjunction with Proposition 6.2 offer two perspectives on machines with memory and nondeterminism. E.g. we shall consider the tensor product of $\mathcal{P}_\omega$ with the stack monad to model push-down automata. As Proposition 6.2 indicates, this monad embeds into the monad with functorial part $TX = \mathcal{P}_\omega(X \times \Gamma^*)^{\Gamma^*}$. On the other hand, by Proposition 6.3, this tensor product is equivalent to a semimodule monad.

Definition 6.4 (Multi-stack nondeterministic $\mathbb{T}$-automaton). A *Multi-stack nondeterministic $\mathbb{T}$-automaton* is a $\mathbb{T}$-automaton ($\bigstar$) for which $\mathbb{T}$ is the tensor of m copies of the stack monad and $\mathcal{P}_\omega$; B is the set of m-ary predicates over Γ^* consisting of all

those $p \in 2^{\Gamma^* \times \cdots \times \Gamma^*}$ for each of which there is a k such that if for any i, $|w_i| \geq k$ then $p(w_1 u_1, \ldots, w_m u_m) = p(w_1 \ldots, w_m)$; and for any $s \in (\Gamma^*)^m$, $f : (\Gamma^*)^m \to \mathcal{P}_\omega(B \times (\Gamma^*)^m) \in TB$ we have $a^{\mathfrak{m}}(f)(s)$ iff $\exists s' \in (\Gamma^*)^m.\ \exists p \in B.\ f(s)(p, s') \wedge p(s')$.

We now obtain the following result.

Theorem 6.5. *For any m let $\mathcal{L}_m$ be the following class of all languages $\{w \in A^* \mid [\![x_0]\!]_{\mathfrak{m}}(w)(\gamma_0, \ldots, \gamma_0)\}$ with $\mathfrak{m}$ ranging over nondeterministic multistack $\mathbb{T}$-automata with m stacks, x_0 ranging over the state space of $\mathfrak{m}$ and γ_0 ranging over Γ. Then $\mathcal{L}_1$ contains exactly context-free languages; for all $m > 2$, $\mathcal{L}_m$ contains exactly nondeterministic linear time languages, i.e. $\mathcal{L}_m = \mathsf{NTIME}(n)$; and $\mathcal{L}_2$ sits properly between $\mathcal{L}_1$ and $\mathcal{L}_3$.*

Theorem 6.5 shows, on the one hand, that the coalgebraic formalization of nondeterministic pushdown automata as nondeterministic $\mathbb{T}$-automata over one stack is adequate in the sense that it recognizes the same class of languages. On the other hand, it indicates the boundaries of the present model: it seems unlikely to capture languages beyond $\mathsf{NTIME}(n)$ (e.g. all recursive ones) by a computationally feasible class of $\mathbb{T}$-automata. This is not surprising in view of the early work on (quasi-)real-time recognizable languages [4], which underlies the proof of Theorem 6.5. We return to this issue in Section 7 where we provide an extension of the present semantics that allows us to capture language classes up to recursively enumerable ones.

We conclude this section with an easy corollary of Theorem 6.5 and Proposition 2.2 contrasting the results in [20, 3].

Corollary 6.6. *Trace equivalence of $\mathbb{T}$-automata is Π^0_1-complete.*

7 CPS-transforms of $\mathbb{T}$-automata and r.e.-languages

Theorem 6.5 suggests that the present trace semantics is unlikely to produce languages beyond $\mathsf{NTIME}(n)$ under a computationally convincing choice of the components of (★). The approach suggested by the classical formal language theory is to replace A with the set $A_\tau = A \cup \{\tau\}$, where τ is a new *unobservable action*, but use the formal power series $A^* \to B$ as the semantic domain instead of $A_\tau^* \to B$. The new *observational semantics* is supposed to be obtainable from the standard one by "eliminating" the unobservable action.

Before we proceed, we introduce an important reformulation of our standard trace equivalence, which is of independent interest. Let $a : TB \to B$ be a $\mathbb{T}$-algebra. We denote by $\mathbb{T}_B$ the continuation monad with $T_B X = B^X \to B$. We can map $\mathbb{T}$ to $\mathbb{T}_B$ by sending any $p : TX$ to $\kappa_X(p) = \lambda f.\, (a \cdot Tf(p)) \in T_B X$. This $\kappa : \mathbb{T} \to \mathbb{T}_B$ is a monad morphism; in fact, it is well known that for any monad $\mathbb{T}$ on a category with powers there is a bijective correspondence between Eilenberg-Moore algebras on B and monad morphisms from $\mathbb{T}$ to $\mathbb{T}_B$ (see e.g. Kock [18, Theorem 3.2]).

Now, given a $\mathbb{T}$-automaton (★), we define a $\mathbb{T}_B$-automaton[1] $m_* : X \to B \times (T_B X)^A$ by $o^{m_*} = o^m$, $t^{m_*} = \kappa_X t^m$, $a^{m_*} = \lambda t.\, t(\mathrm{id})$ where it is easy to see that $a^{m_*} : T_B B \to B$ is a $\mathbb{T}$-algebra. We call this automaton the *CPS-transform*[2] of (★).

Theorem 7.1. *The trace semantics of a $\mathbb{T}$-automaton and of its CPS-transform agree; more precisely, for every $\mathbb{T}$-automaton (★) and state $x \in X$ we have: $[\![x]\!]_m = [\![x]\!]_{m_*}$.*

This theorem implies Proposition 4.3 announced previously in Section 4. Indeed, if B in (★) is finite then, by definition, $T_B X$ is also finite. Thus, the generalized powerset construction performed on the CPS-transform m_* yields a Moore automaton, and hence we obtain the desired result from Proposition 2.4.

We now proceed with the definition of the new semantics.

Definition 7.2 (ω-additive $\mathbb{T}$-automata). A $\mathbb{T}$-automaton (★) is *ω-additive* if B (besides being $\mathbb{T}$-algebra) is an algebra for the countably supported multiset monad.

In other words, for (★) to be ω-additive B needs to be a commutative monoid with infinite summation. We call such a monoid *ω-additive*.

Lemma 7.3. *If B is an ω-additive monoid and a $\mathbb{T}$-algebra then for any X, $T_B X$ is an ω-additive monoid.*

The ω-additive monoid structure on $T_B X$ allows us to define for any given $\mathbb{T}$-automaton over the alphabet A_τ a $\mathbb{T}_B$-automaton over A. To this end, we first form the CPS-transform of the given $\mathbb{T}$-automaton and then use infinite summation to get rid of unobservable actions τ: given a $\mathbb{T}$-automaton $m : X \to B \times (TX)^{A_\tau}$, we construct $m_v : X \to B \times (T_B X)^A$ with $a^{m_v} = a^{m_*} = \lambda t.\, t(\mathrm{id})$ and with t^{m_v}, o^{m_v} defined as

$$t^{m_v}(x_0, a) = \sum_{i=1}^{\infty} \mathsf{do}\ x_1 \leftarrow t^{m_*}(x_0, \tau); \ldots; x_{i-1} \leftarrow t^{m_*}(x_{i-2}, \tau); t^{m_*}(x_{i-1}, a),$$

$$o^{m_v}(x_0) = o^{m_*}(x_0) + \sum_{i=1}^{\infty} \big(\mathsf{do}\ x_1 \leftarrow t^{m_*}(x_0, \tau); \ldots; t^{m_*}(x_{i-1}, \tau)\big)(o^{m_*}).$$

We define the observational trace semantics for m to be the trace semantics for m_v.

Definition 7.4. Given a $\mathbb{T}$-automaton (★) over input alphabet A_τ, let $[\![x]\!]^\tau_m = [\![x]\!]_{m_v}$.

We proceed to define the class of $\mathbb{T}$-automata corresponding to classical Turing machines, for which the introduced observational trace semantics yields precisely all recursively enumerable languages.

Definition 7.5 (Tape $\mathbb{T}$-automaton). A *tape automaton* is a $\mathbb{T}$-automaton (★) for which $\mathbb{T}$ is the tape monad over Γ; B is the set of predicates over $\mathbb{Z} \times \Gamma^{\mathbb{Z}}$ consisting of all those $p \in 2^{\mathbb{Z} \times \Gamma^{\mathbb{Z}}}$ for each of which there is a k such that $p(i, \sigma) = p(i, \sigma')$ and $p(i, \sigma_{+j}) = p(i + j, \sigma)$ if $\sigma =_{i \pm k} \sigma'$; and $a^m : TB \to B$ is given by evaluation as in Definition 5.3.

[1] We abuse terminology here since $\mathbb{T}_B$ is not finitary.

[2] CPS = *continuation-passing style*; we chose the name because the construction is reminiscent of tranforming a functional program into CPS.

It can be shown that tape $\mathbb{T}$-automata over A_τ are equivalent to deterministic 2-tape Turing machines with input alphabet A, where the first tape is a special read-only and right-only tape holding the input word at the beginning of a computation. Thus, we obtain that tape automata represent all the recursively enumerable languages.

Theorem 7.6. *For every tape automaton m over A_τ, Γ with $|\Gamma| \geq 2$ containing a special blank symbol $\square$, and every state $x \in X$ the following language is recursively enumerable: $\{w \in A^* \mid [\![x]\!]^\tau_m(w)(0, \sigma_\square) = 1\}$, where $\sigma_\square$ is the constant function on $\square$. Conversely, every recursively enumerable language can be represented in this way.*

8 Conclusions and Future Work

In this paper, we have presented the first steps towards a coalgebraic Chomsky hierarchy. We have given a coalgebraic account of machines, languages and expressions and presented several results of our theory including a generic Kleene-style theorem (Theorem 4.6). We have also given the first treatment of Turing machines in a coalgebraic setting: the observational trace semantics of tape automata yields precisely the recursively enumerable languages.

There are several possible directions for future work. We plan to derive a sound calculus of reactive expressions extending [3] and explore the boundaries for completeness (by Corollary 6.6 completeness is only possible for specific choices of $\mathbb{T}$); capture further language and complexity classes, such as the context-sensitive languages. Capturing various classes of machines under the umbrella of coalgebra results in standard tools such as bisimulation proof methods becoming available for those classes of machines and their language semantics. Hence, further investigations into such proof principles are of interest.

Acknowledgements. We would like to thank anonymous referees for their suggestions on improving the presentation and for pointing out relevant literature.

References

[1] Haskell 98 Language and Libraries — The Revised Report. Cambridge University Press (2003), also: J. Funct. Prog. **13** (2003)
[2] Adámek, J., Milius, S., Velebil, J.: Iterative algebras at work. Math. Structures Comput. Sci. 16(6), 1085–1131 (2006)
[3] Bonsangue, M.M., Milius, S., Silva, A.: Sound and complete axiomatizations of coalgebraic language equivalence. ACM Trans. Comput. Log. 14(1), 7:1–7:52 (2013)
[4] Book, R.V., Greibach, S.A.: Quasi-realtime languages. Mathematical Systems Theory 4(2), 97–111 (1970)
[5] Brzozowski, J.A.: Derivatives of regular expressions. J. ACM 11(4), 481–494 (1964)
[6] Coumans, D., Jacobs, B.: Scalars, monads, and categories. In: Sadrzadeh, C.H.M., Grefenstette, E. (eds.) Quantum physics and linguistics. A compositional, diagrammatic discourse. pp. 184–216. Oxford University Press (2013)
[7] Courcelle, B.: Fundamental properties of infinite trees. Theoretical Computer Science 25(2), 95 – 169 (1983)
[8] Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. Springer (2009)

[9] Eilenberg, S.: Automata, Languages, and Machines, Pure and Applied Mathematics, vol. A. Academic Press (1974)
[10] Fiore, M.P., Moggi, E., Sangiorgi, D.: A fully abstract model for the π-calculus. Inf. Comput. 179(1), 76–117 (2002)
[11] Freyd, P.: Algebra valued functors in general and tensor products in particular. Colloq. Math. 14, 89–106 (1966)
[12] Goguen, J.A., Thatcher, J.W., Wagner, E.G., Wright, J.B.: Initial algebra semantics and continuous algebras. J. ACM 24(1), 68–95 (Jan 1977)
[13] Goncharov, S.: Trace semantics via generic observations. In: Heckel, R., Milius, S. (eds.) CALCO 2013. LNCS, vol. 8089 (2013)
[14] Hyland, M., Levy, P.B., Plotkin, G.D., Power, J.: Combining algebraic effects with continuations. Theor. Comput. Sci. 375(1-3), 20–40 (2007)
[15] Jacobs, B.: A bialgebraic review of deterministic automata, regular expressions and languages. In: Futatsugi, K., Jouannaud, J.P., Meseguer, J. (eds.) Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday. LNCS, vol. 4060, pp. 375–404 (2006)
[16] Jacobs, B.: Coalgebraic trace semantics for combined possibilitistic and probabilistic systems. Electr. Notes Theor. Comput. Sci. 203(5), 131–152 (2008)
[17] Jacobs, B., Silva, A., Sokolova, A.: Trace semantics via determinization. In: CMCS'12, LNCS, vol. 7399, pp. 109–129. Springer (2012)
[18] Kock, A.: On double dualization monads. Math. Scand. 27, 151–165 (1970)
[19] Mac Lane, S.: Categories for the Working Mathematician. Springer (1971)
[20] Milius, S.: A sound and complete calculus for finite stream circuits. In: Proc. 25th Annual Symposium on Logic in Computer Science (LICS'10). pp. 449–458. IEEE Computer Society (2010)
[21] Moggi, E.: Notions of computation and monads. Inf. Comput. 93, 55–92 (1991)
[22] Myers, R.: Rational Coalgebraic Machines in Varieties: Languages, Completeness and Automatic Proofs. Ph.D. thesis, Imperial College London (2013)
[23] Plotkin, G., Power, J.: Notions of computation determine monads. In: FoSSaCS'02. LNCS, vol. 2303, pp. 342–356. Springer (2002)
[24] Power, J., Shkaravska, O.: From comodels to coalgebras: State and arrays. In: CMCS'04. ENTCS, vol. 106, pp. 297–314 (2004)
[25] Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3(2), 114–125 (Apr 1959)
[26] Rutten, J.: Universal coalgebra: A theory of systems. Theoret. Comput. Sci. 249, 3–80 (2000)
[27] Rutten, J.J.M.M.: Behavioural differential equations: A coinductive calculus of streams, automata, and power series. Theor. Comput. Sci. 308(1-3), 1–53 (2003)
[28] Segala, R.: Modelling and Verification of Randomized Distributed Real-Time Systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)
[29] Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)
[30] Silva, A.: Kleene coalgebra. Ph.D. thesis, Radboud Univ. Nijmegen (2010)
[31] Silva, A., Bonchi, F., Bonsangue, M., Rutten, J.: Generalizing determinization from automata to coalgebras. LMCS 9(1) (2013)
[32] Syme, D., Granicz, A., Cisternino, A.: Expert F#. Apress (2007)
[33] Varacca, D., Winskel, G.: Distributing probability over non-determinism. Math. Struct. Comput. Sci. 16, 87–113 (2006)
[34] Winter, J., Bonsangue, M.M., Rutten, J.J.M.M.: Coalgebraic characterizations of context-free languages. LMCS 9(3) (2013)