A short introduction to the coalgebraic method

Alexandra Silva, Radboud University Nijmegen

This article contains a brief introduction to coalgebra and an overview of a recent proof of correctness for Brzozowski's algorithm which uses coalgebraic techniques. In the discussion section we briefly discuss the most active research lines in the coalgebra community and take a personal outlook at what future might bring for the field.

1. INTRODUCTION

Coalgebra was established in the last decades [Barwise and Moss 1996; Jacobs et al. 1997; Rutten 2000] as a mathematical framework to study dynamical systems from a modular perspective. In a nutshell, the idea behind the coalgebraic framework is that many data structures and automata-like models can be described in a uniform way as instances of a certain abstract mathematical concept. To illustrate in more detail what this means, consider possibly infinite lists over elements of a given set and deterministic finite automata (DFA).

The type of potentially infinite lists is built-in in many functional languages. For instance, in OCaml it is represented by `'a list` and can be instantiated for two cases, namely `[]` for the empty list and `hd :: tl` for the list with head `hd` of type `'a` and tail `tl` of the same type `'a list`. Concrete infinite lists can then be defined using the `let rec` construct:

```
let rec ones = 1 :: ones
let rec alt  = 1 :: 2 :: alt
```

The first example defines the infinite list of ones $1, 1, 1, 1, \ldots$ and the second the alternating sequence $1, 2, 1, 2, \ldots$.

A DFA (over an alphabet A) consists of a set of states S, together with a distinguished initial state and a subset of final states, and a transition function assigning to each state $s \in S$ and letter $a \in A$ a next state $t \in S$.

At first DFA and infinite lists appear to bear little resemblance to one another: for instance, automata have a set of states and in the definition of infinite lists states were never mentioned. But given a list σ what can we observe? We can observe whether σ is empty or not. If it is not empty, we can observe the head element and we are given the rest of the list. So, in fact we can think as the list and all its suffixes as possible states. At each state we can observe an element (or emptiness) and move to the next state (or stop). In a nutshell:

	states	observations	transition dynamics
`list`	$X \subseteq$ `list`	empty: $X \to 1$ or hd: $X \to A$	tl: $X \to X$
DFA	S	isfinal: $S \to 2$	next: $S \to S^A$

Putting the two structures side by side exposes the similarity: both DFA and lists are of the form $(Q, Q \to \mathcal{T}(Q))$ where $\mathcal{T}$ is the type of the dynamics present in the system (both in terms of observations and transitions; in the case of DFA we omit the initial state). In the case of lists $\mathcal{T}(Q) = 1 + A \times X$ whereas in the case of DFA we

have $\mathcal{T}(Q) = 2 \times X^A$, where $+$ denotes disjoint union of sets and $\times$ cartesian product. Objects of the form $(Q, Q \to \mathcal{T}(Q))$ are precisely $\mathcal{T}$-coalgebras. Formally, Q is an object in a category $\mathbb{C}$ and $\mathcal{T}$ is a an endofunctor in the same category. For the purpose of this note we will mostly think of Q as being a set (an element of the category of sets and functions) and $\mathcal{T}$ a polynomial set functor (built from standard set operations, such as cartesian product, disjoint union, etc).

The strength of the coalgebraic framework lies in the fact that the type $\mathcal{T}$ is rich enough to provide a series of canonical notions, including equivalence (coalgebraic bisimulation) and representatives of behaviour (the so-called final coalgebra). For instance, the final coalgebra of $\mathcal{T}(Q) = 2 \times X^A$ is the set of languages $\mathcal{P}(A^*)$ over the alphabet A, which coincides with the usual semantics defined for automata. Coalgebraic bisimulation also coincides with the standard notion of language equivalence.

The observation that canonical notions of behaviour and equivalence can be derived parametric on the type $\mathcal{T}$ formed the start of the coalgebraic method. More recent research has shown that the type $\mathcal{T}$ is also rich enough to derive specification languages, modal logics, axiomatizations, and algorithms [Cîrstea et al. 2011; Silva 2010].

Having a modular perspective on different models has numerous advantages. First, it provides important bridges between models and even research areas. A recent example of this is the development of NetKAT, a specification and programming language for software-defined networks [Foster et al. 2015; Anderson et al. 2014]. The coalgebraic perspective opened the door to connect work on automata theory and network languages, which guided the definition of the language and corresponding semantics. More interestingly, it also yielded a very efficient procedure for deciding equivalence of NetKAT policies. A second advantage is that many notions, techniques, and algorithms can also be developed at a higher level and then instantiated to the concrete example models. One technique of particular importance is coinduction: the proof principle associated with coalgebraic bisimulation. Coinduction is dual to the well-known principle of induction and it provides a powerful reasoning technique to study equivalence of possibly infinite behaviours. Recent years have witnessed a wide range of articles in exploring enhancements of the coinduction proof method [Pous 2008; Rot et al. 2013; Rot et al. 2014; Bonchi and Pous 2015] increasing even further its applicability. A particular beautiful example of the advantages of the coalgebraic perspective is the recent work of Bonchi&Pous [Bonchi and Pous 2015], featured on the January 2015 cover of CACM, studying an enhanced coinduction method to decide language equivalence for non-deterministic finite automata (NFA). The proposed algorithm improved the classical result of Hopcroft&Karp from the 70's. Their work vividly demonstrates that even classical, well-studied problems like NFA equivalence, can still offer surprising research opportunities, and new ideas that may lead to elegant algorithmic improvements of practical importance.

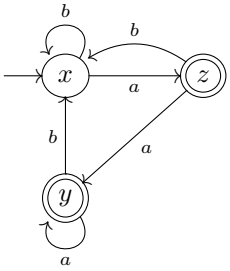
The algorithmic aspects of coalgebra and coinduction have been a quite active recent interest in the community. Understanding existing algorithms and how this abstract perspective can enable development of analogous algorithms for other models has been explored for several algorithms. One such example is Brzozowski's minimization algorithm. In this short note we will explain Brzozowski's algorithm from a coalgebraic perspective and explain how this opens the door to further generalisations without having to re-prove correctness. We will conclude with a discussion on what have been the main lines of research in the coalgebra community and what the future might hold for the field.

2. BRZOZOWSKI'S ALGORITHM, BY EXAMPLE

Brzozowski's minimisation algorithm is one of the simplest minimisation algorithms to explain, though the reason on why it works looks at first sight like magic. Given a (reachable) DFA with state space X the algorithm has four steps.

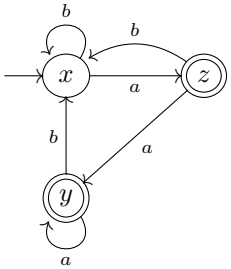
```
  Brzozowski (X)
(1) reverse and determinize;
(2) take the reachable part;
(3) reverse and determinize;
(4) take the reachable part.
```

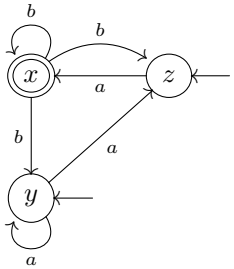
The claim is that the automaton obtained after this procedure is the minimal automaton accepting the same language as the automaton one started with. Let us illustrate the algorithm with an example. Consider the following automaton over the two-letter alphabet $A = \{a, b\}$:



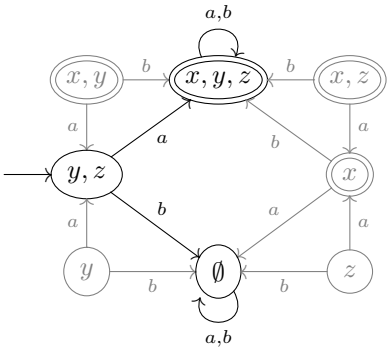
where we denote the final states by double circles and we mark a distinguished initial state with an unlabelled incoming arrow. In the above automaton the state space is $\{x, y, z\}$ where x is the initial state and y and z are final states. It is not too difficult to see that state x accepts all words that end with an a: $L(x) = \{a, b\}^* a$. The above automaton is reachable (all states can be reached from x) but it is not minimal because states y and z accept the same language, namely: $L(y) = \varepsilon + \{a, b\}^* a = L(z)$.

Let us now execute the steps of Brzozowski's algorithm. First, we reverse all transitions and swap initial and final states.

$X =$ 

$rev(X) =$ 

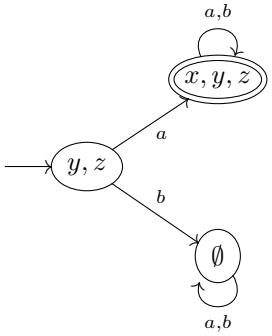
The resulting automaton $rev(X)$ is non-deterministic and hence we construct the corresponding deterministic automaton using the usual subset contruction:



Here, we depict in gray the unreachable states (hence already applying step (2) of the algorithm). The subset construction builds an automaton where the new state space consists of sets of states from the original automaton. The initial state is the set of initial states – $\{y, z\}$ – and a subset V is a final state in the new automaton if $x \in V$ (since x was the only final state in the reversed automaton). Transitions are computed by collecting possible transitions of the non-deterministic automaton:

$$V \xrightarrow{a} W \qquad W = \{w \mid \ v \xrightarrow{a} w \ , v \in V\}$$

All in all, after two steps of the algorithm we end up with the automaton

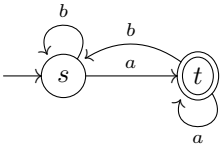


Now note that this new automaton accepts the reverse of the language accepted by X, namely all words starting with an a:

$$L(\, det(rev(X))\,) \ = \ a\,\{a, b\}^* \ = \ reverse(\, L(X)\,)$$

More surprisingly, it is minimal: no two states accept the same language.

Repeating the same two steps on the 3-state automaton above we obtain the following automaton



which is minimal and accepts $\{a, b\}^*\, a$.

Brzozowski's algorithm has attracted some attention in the last couple of years with several papers proposing alternative proofs of correctness and generalisations [Bonchi et al. 2012; Bezhanishvili et al. 2012; Kiefer and Wachter 2014; Bonchi et al. 2014]. This renewed attention was sparked by Prakash Panangaden, SIGLOG's first chair, who during his sabbatical in Oxford gave a nice exposition about a probabilistic extension of the algorithm that he had been working on with his colleagues at McGill [Dinculescu et al. 2011].

Why does the algorithm work? What is the crucial insight that opens the door to further generalisations? We will now briefly describe the ideas behind the proofs presented in [Bonchi et al. 2012; Bezhanishvili et al. 2012].

The main observation in [Bonchi et al. 2012] is that steps `(1)` (and `(3)`) can in fact be captured by using a single well-known categorical construction.

```
Brzozowski (X)
(1) reverse and determinize;
(2) take the reachable part;
(3) reverse and determinize;
(4) take the reachable part.
```

In particular, let $t\colon X \to X^A$ be the transition function of a deterministic automaton. The transition function $\hat{t}$ of the reversed and determinized automaton can be defined as

$$\begin{aligned}&\hat{t}\colon \mathcal{P}(X) \to \mathcal{P}(X)^A\\ &\hat{t}(U)(a) = a - \text{predecessors of } U = \bigcup_{u\in U} \{x \in X \mid t(u)(a) = x\}\end{aligned}$$

Another way to equivalently write this function is by using the characteristic representation of a set.

$$\begin{aligned}&\hat{t}\colon 2^X \to (2^X)^A\\ &\hat{t}(\varphi)(a)(x) = \varphi(t(x)(a))\end{aligned}$$

In fact, this definition can be obtained in 3 steps out of which the first and last are just (un)currying (and hence isomorphisms).

$$\begin{array}{c|c|c|c|c} X & X \times A & & 2^{X\times A} & (2^X)^A \\ {\scriptstyle t}\downarrow & \downarrow & \xmapsto{2^{(-)}} & \uparrow & \uparrow{\scriptstyle \hat{t}} \\ X^A & X & & 2^X & 2^X \end{array}$$

The middle step, which we denote here by $2^{(-)}$ is the application of the *contravariant powerset functor* which is defined as follows, for a set X and a function $f\colon X \to Y$.

$$\begin{aligned}&2^X = \{\varphi \mid \varphi\colon X \to 2\} \cong \{U \mid U \subseteq X\} && 2^f\colon 2^Y \to 2^X\\ &&& 2^f(\varphi) = \varphi \circ f\end{aligned}$$

The core property of $2^{(-)}$ that justifies the correctness of the algorithm is the fact that $2^{(-)}$ maps surjective functions to injective functions. In the algorithm this corresponds to mapping a reachable automaton to an observable automaton (where no two states accept the same language) which will accept the reverse language.

Describing *reverse and determinize* using the contravariant powerset functor is also the crucial observation for generalizations. The above definition of $2^{(-)}$ can for instance trivially be generalised to any output set B:

$$\begin{aligned}&B^X = \{\phi \mid \phi\colon X \to B\} && B^f\colon B^Y \to B^X\\ &&& B^f(\varphi) = \varphi \circ f\end{aligned}$$

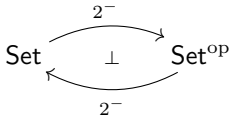
This means that one can now easily describe Brzozowski's algorithm for Moore automata: $X \longrightarrow B \times X^A$. And the important remark is that the proof of correctness does not need to be changed.

This example gives the indication that the type of the coalgebra under study – for DFA $2 \times (-)^A$ and for Moore automata $B \times (-)^A$ – is rich enough also to derive algorithms. Modular derivation of algorithms where both proofs of correctness and complexity analysis can be done from the type of the coalgebra would be a major advance for the field, as we discuss below.

This generalisation to Moore automata might seem mild but in fact it has shown to be extremely useful in various applications stemming from concurrency theory, program verification and software-defined networks. In particular, Moore automata (for an appropriate output set B) are the automata type needed to capture various equivalences of interest in concurrency theory (most of which is included in the van Glabbeek spectrum plus must/may semantics [Bonchi et al. 2012; Bonchi et al. 2013]). Moore automata are also the models corresponding to KAT [Kozen 1997], a program logic of interest for program verification and compiler optimisation, and NetKAT [Foster et al. 2015; Anderson et al. 2014] an extension of KAT to specify and reason about software-defined networks. Interestingly, the Moore automaton functor can also be defined in the category of vector spaces (B has to be a field) and this then yields the so-called *linear weighted automata* [Schützenberger 1961; Boreale 2009] and a corresponding Brzozowski algorithm. Again, the proof of correctness does not need to be changed because the category of vector spaces has the necessary properties that were used from the category of Sets and functors in the proof presented in [Bonchi et al. 2012] (namely epi-mono factorizations).

A more abstract perspective on the correctness of Brzozowski's algorithm was explored in [Bezhanishvili et al. 2012; Bonchi et al. 2014], where dual adjunctions of automata were used in order to recover the algorithm and also prove its correcteness[1].

There is an adjunction between Set, the category of sets and functions, and $\mathsf{Set}^{\mathrm{op}}$ induced by the contravariant powerset functor $2^{(-)} \colon \mathsf{Set} \to \mathsf{Set}^{\mathrm{op}}$:

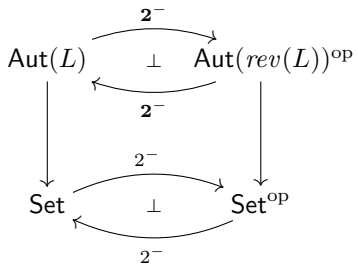


This dual adjunction gives a one-to-one correspondence between morphisms $f \colon X \to 2^Y$ in Set and morphisms $\hat{f} \colon 2^X \to Y$ in $\mathsf{Set}^{\mathrm{op}}$ (that is, $\hat{f} \colon Y \to 2^X$ in Set). This correspondence is given by the following equation

$$y \in f(x) \iff x \in \hat{f}(y)$$

[1] To be precise, in [Bezhanishvili et al. 2012] the authors only explored algebra-coalgebra duality, but we will here describe the related perspective of [Bonchi et al. 2014].

Brzozowski's algorithm is obtained once one observes that the above dual adjunction actual lifts to a suitable category of automata



Starting with an automaton $\mathcal{A}$ accepting a language L (an object in $\mathsf{Aut}(L)$) we obtain an automaton $\mathcal{A}^r = \mathbf{2}(\mathcal{A})$ accepting the reverse language $rev(L)$ (an object in $\mathsf{Aut}(rev(L))$). If $\mathcal{A}$ is a reachable automaton then $\mathbf{2}(\mathcal{A})$ will be an observable automaton (that is, no two states accept the same language). If we now take reachability and apply $\mathbf{2}^- : \mathsf{Set}^{\mathrm{op}} \to \mathsf{Set}$ to obtain an observable automaton accepting $rev(rev(L)) = L$.

Reachability and observability can be described in this setting as universal maps. In a category initial and final objects play an important role. An object $\mathcal{I}$ is initial if there exists a unique map $\mathcal{I} \to O$ into any other object O in the category. Dually, an object $\mathcal{Z}$ is final if there exists a unique map $O \to \mathcal{Z}$ from any other object O into $\mathcal{Z}$. Initial and final objects are intimately connected to definitions of (co)algebraic datatypes in functional languages and also to the notions of induction and coinduction, respectively. We will not explore this here, but instead will use the universal properties of initial and final objects in the category of automata to define reachability and observability.

Let $\mathcal{I}$ and $\mathcal{Z}$ be, respectively, the initial and final objects in $\mathsf{Aut}(L)$ (or $\mathsf{Aut}(rev(L))$) and let $\mathcal{X}$ be an automaton. $\mathcal{X}$ is reachable if the unique morphism from the initial object $r : \mathcal{I} \to \mathcal{X}$ is surjective. $\mathcal{X}$ is observable if the final morphism $o : \mathcal{X} \to \mathcal{Z}$ is injective. The initial object morphism $r : \mathcal{I} \to \mathcal{X}$ is mapped by $\mathbf{2}^-$ to the final morphism $o : \mathbf{2}^{\mathcal{X}} \to \mathcal{Z}$ in $\mathsf{Aut}(rev(L))$. Consequently, if $\mathcal{X}$ is reachable then $\mathcal{A}^r = \mathbf{2}(\mathcal{X})$ is observable.

3. DISCUSSION

Coalgebra has been a very active research area in the last few decades. The dedicated workshop – *Coalgebraic Methods in Computer Science* – was organised for the first time when ETAPS started, in 1998, by Bart Jacobs, Larry Moss, Horst Reichel, and Jan Rutten. Since then, it has always been collocated with ETAPS, becoming bi-annual since the start of CALCO (Conference on Algebra and Coalgebra) in 2005 (CALCO has been organised as independent event, with the exception of the upcoming 2015 edition in which it will be co-located with MFPS XXXI, the 31st edition of the conference on Mathematical Foundations of Programming Semantics). In 2010, there was a special celebration for the 10th edition of CMCS, with four invited talks covering the main areas of research:

— Probabilistic systems coalgebraically (Ana Sokolova)
— Logic and coalgebra (Dirk Pattinson)
— Operational semantics coalgebraically (Bartek Klin)
— Coalgebra in functional programming and type theory (Venanzio Capretta)

After the workshop, all invited speakers wrote overview articles which were published in a special issue [Jacobs et al. 2011]. These four areas have been key in the widespread of coalgebra in other areas like concurrency theory (e.g. [Mislove et al. 2004; Hasuo 2010; Schröder and Venema 2010; Kerstan and König 2012; Madiot et al. 2014; Urabe

and Hasuo 2014]), programming language semantics (e.g. [Bonchi and Pous 2013; Abel et al. 2013; Staton and Levy 2013; Jeannin et al. 2013; Bonchi et al. 2015; Foster et al. 2015; Staton 2015; Anderson et al. 2014]), and artificial intelligence (e.g. [Goré et al. 2010; Schröder and Pattinson 2011; Kulacka et al. 2013]).

In the last five years, there has been a large body of research in applying (co)algebraic methods to automata theory (e.g. [Bonsangue et al. 2013; Winter et al. 2013; Rutten et al. 2013; Silva et al. 2013; Adámek et al. 2014a; Adámek et al. 2014b; Goncharov et al. 2014]). One thing that became evident in this line of research was that the combination of coalgebraic and algebraic methods is essential in order to fully understand automata constructions and algorithms. In my personal opinion, the synergy between algebra and coalgebra has a lot of potential to generate new results and algorithmic insights. I think that a framework based solely on just either coalgebra or algebra will not explore the full potential of abstraction offered by both frameworks.

Apart from the applications in automata theory I think there are two directions that show promising challenges for the theory of coalgebras and in which I believe coalgebra could have significant impact.

The first direction concerns the applications of coinduction in the context of verification and, in particular, quantitative verification. To achieve the verification of real-world systems, the study of efficient coinductive techniques for specifying and reasoning about dynamical systems is needed. A system can be expressed as a coalgebra, and its desired observable behaviour can be expressed as a (generalised) regular expression. The correctness of the system can then be verified by proving that these two formalisms are observably equivalent or, more realistically, that one is a refinement of the other. Coalgebra has mainly focused on techniques for equivalence, and lifting the general theory to handle simulation/refinement is a pressing research task that has received recent interest [Hasuo 2010; Urabe and Hasuo 2014] but is far from finished. One needs efficient algorithms for the calculation of a witness of the equivalence or refinement, which in the coalgebraic approach consists of a suitable (bi)simulation relation. Already for ordinary regular expressions, a naive implementation can be exponentially complex, even if most of the operations are polynomial. However, very efficient algorithms exist to compositionally derive a finite automaton from ordinary regular expressions. Extending these algorithms to the generalised algebraic and coalgebraic framework of regular expressions and implement them in existing tools would be an important step in enabling the application to industrial case studies. In recent work on NetKAT [Anderson et al. 2014], an instance of a generalised bisimulation procedure provides an efficient algorithm for a specification language for software-defined networking (SDN), which brings coalgebra closer to a more realistic application domain in specification and verification.

In the context of this first line of research, having the right notion of equivalence is crucial. Defining the behaviour or the behavioural similarity of two systems that behave the same has been a fundamental issue in computer science. Many different equivalences, usually referred as behavioural equivalences, have been introduced for different kinds of systems (e.g., functional, non-deterministic and concurrent systems). A huge corpus of conceptual and computational tools consisting of algorithms, proof techniques, rule formats, and languages have been developed to reason about systems and their behavioural equivalences.

The growing interest in quantitative properties has motivated the definition of quantitative models and related notions of behavioural equivalences. For many quantitative properties, standard notions of equivalence are inadequate since they only allow Boolean reasoning: either two systems are equivalent or they are not. As an example, consider a system consisting of a producer, a consumer, and a buffer connecting them: whenever the buffer is full, the newly-produced resources get lost. Suppose that the

producer or the consumer are complex networks, and suppose that we know by empirical data their rates of production and consumption (examples of this kind are widely studied in the performance modelling of systems). A probabilistic model of this system allows for several interesting quantitative questions, like: How far is the system with a buffer of capacity n from the ideal system that never loses resources? For which values of n is the distance between the behaviours of these two systems smaller than a threshold ε? Motivated by these considerations, many authors have recently proposed several behavioural metrics or closely related notions [Desharnais et al. 1999; van Breugel and Worrell 2006; Cerný et al. 2010; Bacci et al. 2013]. Unlike ordinary equivalences, behavioural metrics express the distance between the behaviours of systems: if the distance is zero, then the systems are behaviourally equivalent. These works show the effectiveness of behavioural metrics as foundations of quantitative reasoning, but there is much work still to be done. In particular, many conceptual and computational tools are not yet available for metrics. There is a pressing need of lifting such tools from equivalences to metrics, which I think can be achieved in a modular way using the theory of coalgebras.

The second direction is in the area of automata learning. Learning techniques have become increasingly important for their applications to a wide variety of software engineering problems, especially in the analysis and testing of complex systems. Recently, they have been successfully applied to security protocol testing, the analysis of botnet command and control protocols, in regression testing of telecommunication protocols, and conformance testing of communication protocols. For each application domain, the type of automaton to be learned varies, hence the algorithm needs to be adapted and this generates challenging problems. For all the proposed algorithms, which followed the seminal work of Angluin in 1987 [Angluin 1987], all the correctness and complexity results need to be re-proved for every different case, even if the model under study is not a major variation on deterministic automata. Similar observations apply to the area of Bayesian or probabilistic learning, where the development of algorithms is done on a call-by-need basis. I believe that coalgebra provides the perfect framework for the systematic study and development of learning algorithms in automata and Bayesian networks. Recent preliminary work [Jacobs and Silva 2014] shows the feasibility and the potential of the approach. There are a lot of challenges to be tackled, which is also a source of inspiration for further research in the coalgebra community, but the point in the horizon is to provide a strong mathematical foundation and bring compositional techniques to learning algorithms.

REFERENCES

Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. 2013. Copatterns: Programming Infinite Structures by Observations. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '13)*. ACM, New York, NY, USA, 27–38. DOI:http://dx.doi.org/10.1145/2429069.2429075

Jirí Adámek, Stefan Milius, Robert S. R. Myers, and Henning Urbat. 2014a. Generalized Eilenberg Theorem I: Local Varieties of Languages. In *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings (Lecture Notes in Computer Science)*, Anca Muscholl (Ed.), Vol. 8412. Springer, 366–380. DOI:http://dx.doi.org/10.1007/978-3-642-54830-7_24

Jirí Adámek, Robert S. R. Myers, Henning Urbat, and Stefan Milius. 2014b. On Continuous Nondeterminism and State Minimality. *Electr. Notes Theor. Comput. Sci.* 308 (2014), 3–23. DOI:http://dx.doi.org/10.1016/j.entcs.2014.10.002

Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. 2014. NetkAT: semantic foundations for networks. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego,*

CA, USA, January 20-21, 2014, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 113–126. DOI:http://dx.doi.org/10.1145/2535838.2535862

Dana Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75, 2 (1987), 87–106. DOI:http://dx.doi.org/10.1016/0890-5401(87)90052-6

Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. 2013. On-the-Fly Exact Computation of Bisimilarity Distances. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings (Lecture Notes in Computer Science)*, Nir Piterman and Scott A. Smolka (Eds.), Vol. 7795. Springer, 1–15. DOI:http://dx.doi.org/10.1007/978-3-642-36742-7_1

Paolo Baldan and Daniele Gorla (Eds.). 2014. *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*. Lecture Notes in Computer Science, Vol. 8704. Springer. DOI:http://dx.doi.org/10.1007/978-3-662-44584-6

Jon Barwise and Larry Moss. 1996. *Vicious Circles*. Center for the Study of Language and Information - Stanford.

Nick Bezhanishvili, Clemens Kupke, and Prakash Panangaden. 2012. Minimization via Duality. In *Logic, Language, Information and Computation - 19th International Workshop, WoLLIC 2012, Buenos Aires, Argentina, September 3-6, 2012. Proceedings (Lecture Notes in Computer Science)*, C.-H. Luke Ong and Ruy J. G. B. de Queiroz (Eds.), Vol. 7456. Springer, 191–205. DOI:http://dx.doi.org/10.1007/978-3-642-32621-9_14

Filippo Bonchi, Marcello M. Bonsangue, Georgiana Caltais, Jan J. M. M. Rutten, and Alexandra Silva. 2012. Final Semantics for Decorated Traces. *Electr. Notes Theor. Comput. Sci.* 286 (2012), 73–86. DOI:http://dx.doi.org/10.1016/j.entcs.2012.08.006

Filippo Bonchi, Marcello M. Bonsangue, Helle Hvid Hansen, Prakash Panangaden, Jan J. M. M. Rutten, and Alexandra Silva. 2014. Algebra-coalgebra duality in Brzozowski's minimization algorithm. *ACM Trans. Comput. Log.* 15, 1 (2014), 3. DOI:http://dx.doi.org/10.1145/2490818

Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. 2012. Brzozowski's Algorithm (Co)Algebraically. In *Logic and Program Semantics - Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday (Lecture Notes in Computer Science)*, Robert L. Constable and Alexandra Silva (Eds.), Vol. 7230. Springer, 12–23. DOI:http://dx.doi.org/10.1007/978-3-642-29485-3_2

Filippo Bonchi, Georgiana Caltais, Damien Pous, and Alexandra Silva. 2013. Brzozowski's and Up-To Algorithms for Must Testing. In *Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings (Lecture Notes in Computer Science)*, Chung-chieh Shan (Ed.), Vol. 8301. Springer, 1–16. DOI:http://dx.doi.org/10.1007/978-3-319-03542-0_1

Filippo Bonchi and Damien Pous. 2013. Checking NFA equivalence with bisimulations up to congruence, See Giacobazzi and Cousot [2013], 457–468. DOI:http://dx.doi.org/10.1145/2429069.2429124

Filippo Bonchi and Damien Pous. 2015. Hacking nondeterminism with induction and coinduction. *Commun. ACM* 58, 2 (2015), 87–95. DOI:http://dx.doi.org/10.1145/2713167

Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. 2015. Full Abstraction for Signal Flow Graphs, See Rajamani and Walker [2015], 515–526. DOI:http://dx.doi.org/10.1145/2676726.2676993

Marcello M. Bonsangue, Stefan Milius, and Alexandra Silva. 2013. Sound and Complete Axiomatizations of Coalgebraic Language Equivalence. *ACM Trans. Comput. Log.* 14, 1 (2013), 7. DOI:http://dx.doi.org/10.1145/2422085.2422092

Michele Boreale. 2009. Weighted Bisimulation in Linear Algebraic Form. In *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings (Lecture Notes in Computer Science)*, Mario Bravetti and Gianluigi Zavattaro (Eds.), Vol. 5710. Springer, 163–177. DOI:http://dx.doi.org/10.1007/978-3-642-04081-8_12

Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. 2010. Simulation Distances, See Gastin and Laroussinie [2010], 253–268. DOI:http://dx.doi.org/10.1007/978-3-642-15375-4_18

Corina Cîrstea, Alexander Kurz, Dirk Pattinson, Lutz Schröder, and Yde Venema. 2011. Modal Logics are Coalgebraic. *Comput. J.* 54, 1 (2011), 31–41. DOI:http://dx.doi.org/10.1093/comjnl/bxp004

Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. 1999. Metrics for Labeled Markov Systems. In *CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings (Lecture Notes in Computer Science)*, Jos C. M. Baeten and Sjouke Mauw (Eds.), Vol. 1664. Springer, 258–273. DOI:http://dx.doi.org/10.1007/3-540-48320-9_19

Monica Dinculescu, Christopher Hundt, Prakash Panangaden, Joelle Pineau, and Doina Precup. 2011. The Duality of State and Observation in Probabilistic Transition Systems. In *Logic, Language, and Computation - 9th International Tbilisi Symposium on Logic, Language, and Computation, TbiLLC 2011, Kutaisi, Georgia, September 26-30, 2011, Revised Selected Papers (Lecture Notes in Computer Science)*,

Guram Bezhanishvili, Sebastian Löbner, Vincenzo Marra, and Frank Richter (Eds.), Vol. 7758. Springer, 206–230. DOI:http://dx.doi.org/10.1007/978-3-642-36976-6_14

Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. 2015. A Coalgebraic Decision Procedure for NetKAT, See Rajamani and Walker [2015], 343–355. DOI:http://dx.doi.org/10.1145/2676726.2677011

Paul Gastin and François Laroussinie (Eds.). 2010. *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*. Lecture Notes in Computer Science, Vol. 6269. Springer. DOI:http://dx.doi.org/10.1007/978-3-642-15375-4

Roberto Giacobazzi and Radhia Cousot (Eds.). 2013. *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*. ACM. http://dl.acm.org/citation.cfm?id=2429069

Sergey Goncharov, Stefan Milius, and Alexandra Silva. 2014. Towards a Coalgebraic Chomsky Hierarchy - (Extended Abstract). In *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings (Lecture Notes in Computer Science)*, Josep Diaz, Ivan Lanese, and Davide Sangiorgi (Eds.), Vol. 8705. Springer, 265–280. DOI:http://dx.doi.org/10.1007/978-3-662-44602-7_21

Rajeev Goré, Clemens Kupke, Dirk Pattinson, and Lutz Schröder. 2010. Global Caching for Coalgebraic Description Logics. In *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings (Lecture Notes in Computer Science)*, Jürgen Giesl and Reiner Hähnle (Eds.), Vol. 6173. Springer, 46–60. DOI:http://dx.doi.org/10.1007/978-3-642-14203-1_5

Ichiro Hasuo. 2010. Generic Forward and Backward Simulations II: Probabilistic Simulation, See Gastin and Laroussinie [2010], 447–461. DOI:http://dx.doi.org/10.1007/978-3-642-15375-4_31

Bart Jacobs, Milad Niqui, Jan J. M. M. Rutten, and Alexandra Silva. 2011. Preface. *Theor. Comput. Sci.* 412, 38 (2011), 4967–4968. DOI:http://dx.doi.org/10.1016/j.tcs.2011.06.001

Bart Jacobs, JJMM Rutten, and others. 1997. An introduction to (co) algebra and (co) induction. *EATCS Bulletin* 62 (1997), 222–259.

Bart Jacobs and Alexandra Silva. 2014. Automata Learning: A Categorical Perspective. In *Horizons of the Mind. A Tribute to Prakash Panangaden - Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday (Lecture Notes in Computer Science)*, Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten (Eds.), Vol. 8464. Springer, 384–406. DOI:http://dx.doi.org/10.1007/978-3-319-06880-0_20

Jean-Baptiste Jeannin, Dexter Kozen, and Alexandra Silva. 2013. Language Constructs for Non-Well-Founded Computation. In *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings (Lecture Notes in Computer Science)*, Matthias Felleisen and Philippa Gardner (Eds.), Vol. 7792. Springer, 61–80. DOI:http://dx.doi.org/10.1007/978-3-642-37036-6_4

Henning Kerstan and Barbara König. 2012. Coalgebraic Trace Semantics for Probabilistic Transition Systems Based on Measure Theory. In *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings (Lecture Notes in Computer Science)*, Maciej Koutny and Irek Ulidowski (Eds.), Vol. 7454. Springer, 410–424. DOI:http://dx.doi.org/10.1007/978-3-642-32940-1_29

Stefan Kiefer and Björn Wachter. 2014. Stability and Complexity of Minimising Probabilistic Automata. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II (Lecture Notes in Computer Science)*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.), Vol. 8573. Springer, 268–279. DOI:http://dx.doi.org/10.1007/978-3-662-43951-7_23

Dexter Kozen. 1997. Kleene Algebra with Tests. *ACM Trans. Program. Lang. Syst.* 19, 3 (1997), 427–443. DOI:http://dx.doi.org/10.1145/256167.256195

Agnieszka Kulacka, Dirk Pattinson, and Lutz Schröder. 2013. Syntactic Labelled Tableaux for Lukasiewicz Fuzzy ALC. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, Francesca Rossi (Ed.). IJCAI/AAAI. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6831

Jean-Marie Madiot, Damien Pous, and Davide Sangiorgi. 2014. Bisimulations Up-to: Beyond First-Order Transition Systems, See Baldan and Gorla [2014], 93–108. DOI:http://dx.doi.org/10.1007/978-3-662-44584-6_8

Michael W. Mislove, Joël Ouaknine, Dusko Pavlovic, and James Worrell. 2004. Duality for Labelled Markov Processes. In *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Held as Part of the Joint European Conferences on Theory*

and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings (Lecture Notes in Computer Science), Igor Walukiewicz (Ed.), Vol. 2987. Springer, 393–407. DOI:http://dx.doi.org/10.1007/978-3-540-24727-2_28

Damien Pous. 2008. *Techniques modulo pour les bisimulations*. Ph.D. Dissertation. PhD thesis, ENS Lyon.

Sriram K. Rajamani and David Walker (Eds.). 2015. *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. ACM. http://dl.acm.org/citation.cfm?id=2676726

Jurriaan Rot, Filippo Bonchi, Marcello Bonsangue, Damien Pous, JJMM Rutten, and Alexandra Silva. 2014. Enhanced coalgebraic bisimulation. *Mathematical Structures in Computer Science (to appear, 2014)* (2014).

Jurriaan Rot, Marcello M. Bonsangue, and Jan J. M. M. Rutten. 2013. Coalgebraic Bisimulation-Up-To. In *SOFSEM 2013: Theory and Practice of Computer Science, 39th International Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 26-31, 2013. Proceedings (Lecture Notes in Computer Science)*, Peter van Emde Boas, Frans C. A. Groen, Giuseppe F. Italiano, Jerzy R. Nawrocki, and Harald Sack (Eds.), Vol. 7741. Springer, 369–381. DOI:http://dx.doi.org/10.1007/978-3-642-35843-2_32

Jan Rutten, Adolfo Ballester-Bolinches, and Enric Cosme-Llópez. 2013. Varieties and Covarieties of Languages (Extended Abstract). *Electr. Notes Theor. Comput. Sci.* 298 (2013), 7–28. DOI:http://dx.doi.org/10.1016/j.entcs.2013.09.005

Jan J. M. M. Rutten. 2000. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.* 249, 1 (2000), 3–80. DOI:http://dx.doi.org/10.1016/S0304-3975(00)00056-6

Lutz Schröder and Dirk Pattinson. 2011. Description Logics and Fuzzy Probability. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, Toby Walsh (Ed.). IJCAI/AAAI, 1075–1081. http://ijcai.org/papers11/Papers/IJCAI11-184.pdf

Lutz Schröder and Yde Venema. 2010. Flat Coalgebraic Fixed Point Logics, See Gastin and Laroussinie [2010], 524–538. DOI:http://dx.doi.org/10.1007/978-3-642-15375-4_36

Marcel Paul Schützenberger. 1961. On the Definition of a Family of Automata. *Information and Control* 4, 2-3 (1961), 245–270. DOI:http://dx.doi.org/10.1016/S0019-9958(61)80020-X

Alexandra Silva. 2010. *Kleene Coalgebra*. Ph.D. Dissertation. Radboud University Nijmegen.

Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. 2013. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science* 9, 1 (2013). DOI:http://dx.doi.org/10.2168/LMCS-9(1:9)2013

Sam Staton. 2015. Algebraic Effects, Linearity, and Quantum Programming Languages, See Rajamani and Walker [2015], 395–406. DOI:http://dx.doi.org/10.1145/2676726.2676999

Sam Staton and Paul Blain Levy. 2013. Universal properties of impure programming languages, See Giacobazzi and Cousot [2013], 179–192. DOI:http://dx.doi.org/10.1145/2429069.2429091

Natsuki Urabe and Ichiro Hasuo. 2014. Generic Forward and Backward Simulations III: Quantitative Simulations by Matrices, See Baldan and Gorla [2014], 451–466. DOI:http://dx.doi.org/10.1007/978-3-662-44584-6_31

Franck van Breugel and James Worrell. 2006. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comput. Sci.* 360, 1-3 (2006), 373–385. DOI:http://dx.doi.org/10.1016/j.tcs.2006.05.021

Joost Winter, Marcello M. Bonsangue, and Jan J. M. M. Rutten. 2013. Coalgebraic Characterizations of Context-Free Languages. *Logical Methods in Computer Science* 9, 3 (2013). DOI:http://dx.doi.org/10.2168/LMCS-9(3:14)2013