 Foundations of Software Science and Computation Structures

23rd International Conference, FOSSACS 2020
Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020
Dublin, Ireland, April 25–30, 2020, Proceedings
More information about this series at http://www.springer.com/series/7407
ETAPS Foreword

Welcome to the 23rd ETAPS! This is the first time that ETAPS took place in Ireland in its beautiful capital Dublin.

ETAPS 2020 was the 23rd instance of the European Joint Conferences on Theory and Practice of Software. ETAPS is an annual federated conference established in 1998, and consists of four conferences: ESOP, FASE, FoSSaCS, and TACAS. Each conference has its own Program Committee (PC) and its own Steering Committee (SC). The conferences cover various aspects of software systems, ranging from theoretical computer science to foundations of programming language developments, analysis tools, and formal approaches to software engineering. Organizing these conferences in a coherent, highly synchronized conference program enables researchers to participate in an exciting event, having the possibility to meet many colleagues working in different directions in the field, and to easily attend talks of different conferences. On the weekend before the main conference, numerous satellite workshops took place that attracted many researchers from all over the globe. Also, for the second time, an ETAPS Mentoring Workshop was organized. This workshop is intended to help students early in the program with advice on research, career, and life in the fields of computing that are covered by the ETAPS conference.

ETAPS 2020 received 424 submissions in total, 129 of which were accepted, yielding an overall acceptance rate of 30.4%. I thank all the authors for their interest in ETAPS, all the reviewers for their reviewing efforts, the PC members for their contributions, and in particular the PC (co-)chairs for their hard work in running this entire intensive process. Last but not least, my congratulations to all authors of the accepted papers!

ETAPS 2020 featured the unifying invited speakers Scott Smolka (Stony Brook University) and Jane Hillston (University of Edinburgh) and the conference-specific invited speakers (ESOP) Işıl Dillig (University of Texas at Austin) and (FASE) Willem Visser (Stellenbosch University). Invited tutorials were provided by Erika Ábrahám (RWTH Aachen University) on the analysis of hybrid systems and Madhusudan Parthasarathy (University of Illinois at Urbana-Champaign) on combining Machine Learning and Formal Methods. On behalf of the ETAPS 2020 attendants, I thank all the speakers for their inspiring and interesting talks!

ETAPS 2020 took place in Dublin, Ireland, and was organized by the University of Limerick and Lero. ETAPS 2020 is further supported by the following associations and societies: ETAPS e.V., EATCS (European Association for Theoretical Computer Science), EAPLS (European Association for Programming Languages and Systems), and EASST (European Association of Software Science and Technology). The local organization team consisted of Tiziana Margaria (general chair, UL and Lero), Vasileios Koutavas (Lero@UCD), Anila Mjeda (Lero@UL), Anthony Ventresque (Lero@UCD), and Petros Stratis (Easy Conferences).
The ETAPS Steering Committee (SC) consists of an Executive Board, and representatives of the individual ETAPS conferences, as well as representatives of EATCS, EAPLS, and EASST. The Executive Board consists of Holger Hermanns (Saarbrücken), Marieke Huisman (chair, Twente), Joost-Pieter Katoen (Aachen and Twente), Jan Kofron (Prague), Gerald Lüttgen (Bamberg), Tarmo Uustalu (Reykjavik and Tallinn), Caterina Urban (Inria, Paris), and Lenore Zuck (Chicago).

Other members of the SC are: Armin Biere (Linz), Jordi Cabot (Barcelona), Jean Goubault-Larrecq (Cachan), Jan-Friso Groote (Eindhoven), Esther Guerra (Madrid), Jurriaan Hage (Utrecht), Reiko Heckel (Leicester), Panagiotis Katsaros (Thessaloniki), Stefan Kiefer (Oxford), Barbara König (Duisburg), Fabrice Kordon (Paris), Jan Kretinsky (Munich), Kim G. Larsen (Aalborg), Tiziana Margaria (Limerick), Peter Müller (Zurich), Catuscia Palamidessi (Palaiseau), Dave Parker (Birmingham), Andrew M. Pitts (Cambridge), Peter Ryan (Luxembourg), Don Sannella (Edinburgh), Bernhard Steffen (Dortmund), Mariëlle Stoelinga (Twente), Gabriele Taentzer (Marburg), Christine Tasson (Paris), Peter Thiemann (Freiburg), Jan Vitek (Prague), Heike Wehrheim (Paderborn), Anton Wijs (Eindhoven), and Nobuko Yoshida (London).

I would like to take this opportunity to thank all speakers, attendants, organizers of the satellite workshops, and Springer for their support. I hope you all enjoyed ETAPS 2020. Finally, a big thanks to Tiziana and her local organization team for all their enormous efforts enabling a fantastic ETAPS in Dublin!

February 2020

Marieke Huisman
ETAPS SC Chair
ETAPS e.V. President
Preface

This volume contains the papers presented at the 23rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS), which took place in Dublin, Ireland, during April 27–30, 2020. The conference series is dedicated to foundational research with a clear significance for software science. It brings together research on theories and methods to support the analysis, integration, synthesis, transformation, and verification of programs and software systems.

This volume contains 31 contributed papers selected from 98 full paper submissions, and also a paper accompanying an invited talk by Scott Smolka (Stony Brook University, USA). Each submission was reviewed by at least three Program Committee members, with the help of external reviewers, and the final decisions took into account the feedback from a rebuttal phase. The conference submissions were managed using the EasyChair conference system, which was also used to assist with the compilation of these proceedings.

We wish to thank all the authors who submitted papers to FoSSaCS 2020, the Program Committee members, the Steering Committee members and the external reviewers. In addition, we are grateful to the ETAPS 2020 Organization for providing an excellent environment for FoSSaCS 2020 alongside the other ETAPS conferences and workshops.

February 2020

Jean Goubault-Larrecq
Barbara König
Organization

Program Committee

Parosh Aziz Abdulla  Uppsala University, Sweden
Thorsten Altenkirch  University of Nottingham, UK
Paolo Baldan  Università di Padova, Italy
Nick Benton  Facebook, UK
Frédéric Blanqui  Inria and LSV, France
Michele Boreale  Università di Firenze, Italy
Corina Cirstea  University of Southampton, UK
Pedro R. D’Argenio  Universidad Nacional de Córdoba, CONICET, Argentina
Josée Desharnais  Université Laval, Canada
Jean Goubault-Larrecq  Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, Cachan, France
Ichiro Hasuo  National Institute of Informatics, Japan
Delia Kesner  IRIF, Université de Paris, France
Shankara Narayanan Krishna  IIT Bombay, India
Barbara König  Universität Duisburg-Essen, Germany
Sławomir Lasota  University of Warsaw, Poland
Xavier Leroy  Collège de France and Inria, France
Leonid Libkin  University of Edinburgh, UK, and ENS Paris, France
Jean-Yves Marion  LORIA, Université de Lorraine, France
Dominique Méry  LORIA, Université de Lorraine, France
Matteo Mio  LIP, CNRS, ENS Lyon, France
Andrzej Murawski  University of Oxford, UK
Prakash Panangaden  McGill University, Canada
Amr Sabry  Indiana University Bloomington, USA
Lutz Schröder  Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
Sebastian Siebertz  Universität Bremen, Germany
Benoît Valiron  LRI, CentraleSupélec, Université Paris-Saclay, France

Steering Committee

Andrew Pitts (Chair)  University of Cambridge, UK
Christel Baier  Technische Universität Dresden, Germany
Lars Birkedal  Aarhus University, Denmark
Ugo Dal Lago  Università degli Studi di Bologna, Italy
Javier Esparza  
Technische Universität München, Germany

Anca Muscholl  
LaBRI, Université de Bordeaux, France

Frank Pfennig  
Carnegie Mellon University, USA

### Additional Reviewers

Accattoli, Beniamino  
Dell’Erba, Daniele

Alvim, Mario S.  
Deng, Yuxin

André, Étienne  
Eickmeyer, Kord

Argyros, George  
Exibard, Leo

Arun-Kumar, S.  
Faggian, Claudia

Ayala-Rincon, Mauricio  
Fijalkow, Nathanaël

Bacci, Giorgio  
Filali-Amine, Mamoun

Bacci, Giovanni  
Francalanza, Adrian

Balabonski, Thibaut  
Frutos Escrig, David

Basile, Davide  
Galletta, Letterio

Berger, Martin  
Ganian, Robert

Bernardi, Giovanni  
Garrigue, Jacques

Bisping, Benjamin  
Gastin, Paul

Bodeveix, Jean-Paul  
Genaim, Samir

Bollig, Benedikt  
Genest, Blaise

Bonchi, Filippo  
Ghica, Dan

Bonelli, Eduardo  
Goncharov, Sergey

Boulimé, Sylvain  
Gorla, Daniele

Bourke, Timothy  
Guerrini, Stefano

Bradfield, Julian  
Hirschowitz, Tom

Breuvart, Flavien  
Hofman, Piotr

Bruni, Roberto  
Hoshino, Naohiko

Bruse, Florian  
Howar, Falk

Caprriott, Paolo  
Inverso, Omar

Carette, Jacques  
Iván, Szabolcs

Carette, Titouan  
Jaax, Stefan

Carton, Olivier  
Jeandel, Emmanuel

Cassano, Valentin  
Johnson, Michael

Chadha, Rohit  
Kahrs, Stefan

Charguéraud, Arthur  
Kamburjan, Eduard

Cho, Kenta  
Katsumata, Shin-Ya

Choudhury, Vikraman  
Kerjean, Marie

Ciancia, Vincenzo  
Kiefer, Stefan

Clemente, Lorenzo  
Komorida, Yuichi

Colacito, Almudena  
Kop, Cynthia

Corradini, Andrea  
Kremer, Steve

Czerwiński, Wojciech  
Kuperberg, Denis

de Haan, Ronald  
Křetínský, Jan

de Visser, Marc  
Laarman, Alfons
# Contents

Neural Flocking: MPC-Based Supervised Learning of Flocking Controllers .......................... 1
   Usama Mehmood, Shovik Roy, Radu Grosu, Scott A. Smolka, Scott D. Stoller, and Ashish Tiwari

On Well-Founded and Recursive Coalgebras ......................................................... 17
   Jiří Adámek, Stefan Milius, and Lawrence S. Moss

Timed Negotiations ............................................................... 37
   S. Akshay, Blaise Genest, Loïc Hélouët, and Sharvik Mital

Cartesian Difference Categories ...................................................... 57
   Mario Alvarez-Picallo and Jean-Simon Pacaud Lemay

Contextual Equivalence for Signal Flow Graphs .............................................. 77
   Filippo Bonchi, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi

Parameterized Synthesis for Fragments of First-Order Logic Over
Data Words ................................................................................ 97
   Béatrice Béroud, Benedikt Bollig, Mathieu Lehaut, and Nathalie Sznajder

Controlling a Random Population ...................................................................... 119
   Thomas Colcombet, Nathanaël Fijalkow, and Pierre Ohlmann

Decomposing Probabilistic Lambda-Calculi ..................................................... 136
   Ugo Dal Lago, Giulio Guerrieri, and Willem Heijltjes

On the k-synchronizability of Systems ......................................................... 157
   Cinzia Di Giusto, Paëtina Laversa, and Etienne Lozes

General Supervised Learning as Change Propagation with Delta Lenses ............ 177
   Zinovy Diskin

Non-idempotent Intersection Types in Logical Form .................................... 198
   Thomas Ehrhard

On Computability of Data Word Functions Defined by Transducers ............... 217
   Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier

Minimal Coverability Tree Construction Made Complete and Efficient ............ 237
   Alain Finkel, Serge Haddad, and Igor Khmelevsky
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructing Infinitary Quotient-Inductive Types.</td>
<td>257</td>
</tr>
<tr>
<td>Marcelo P. Fiore, Andrew M. Pitts, and S. C. Steenkamp</td>
<td></td>
</tr>
<tr>
<td>Relative Full Completeness for Bicategorical Cartesian Closed Structure</td>
<td>277</td>
</tr>
<tr>
<td>Marcelo Fiore and Philip Saville</td>
<td></td>
</tr>
<tr>
<td>A Duality Theoretic View on Limits of Finite Structures</td>
<td>299</td>
</tr>
<tr>
<td>Mai Gehrke, Tomáš Jakl, and Luca Reggio</td>
<td></td>
</tr>
<tr>
<td>Correctness of Automatic Differentiation via Diffeologies and Categorical Gluing</td>
<td>319</td>
</tr>
<tr>
<td>Mathieu Huot, Sam Staton, and Matthijs Vákár</td>
<td></td>
</tr>
<tr>
<td>Deep Induction: Induction Rules for (Truly) Nested Types</td>
<td>339</td>
</tr>
<tr>
<td>Patricia Johann and Andrew Polonsky</td>
<td></td>
</tr>
<tr>
<td>Exponential Automatic Amortized Resource Analysis</td>
<td>359</td>
</tr>
<tr>
<td>David M. Kahn and Jan Hoffmann</td>
<td></td>
</tr>
<tr>
<td>Concurrent Kleene Algebra with Observations: From Hypotheses to Completeness</td>
<td>381</td>
</tr>
<tr>
<td>Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi</td>
<td></td>
</tr>
<tr>
<td>Graded Algebraic Theories</td>
<td>401</td>
</tr>
<tr>
<td>Satoshi Kura</td>
<td></td>
</tr>
<tr>
<td>A Curry-style Semantics of Interaction: From Untyped to Second-Order Lazy $\lambda\mu$-Calculus</td>
<td>422</td>
</tr>
<tr>
<td>James Laird</td>
<td></td>
</tr>
<tr>
<td>An Axiomatic Approach to Reversible Computation</td>
<td>442</td>
</tr>
<tr>
<td>Ivan Lanese, Iain Phillips, and Irek Ulidowski</td>
<td></td>
</tr>
<tr>
<td>An Auxiliary Logic on Trees: on the Tower-Hardness of Logics Featuring Reachability and Submodel Reasoning</td>
<td>462</td>
</tr>
<tr>
<td>Alessio Mansutti</td>
<td></td>
</tr>
<tr>
<td>The Inconsistent Labelling Problem of Stutter-Preserving Partial-Order Reduction</td>
<td>482</td>
</tr>
<tr>
<td>Thomas Neele, Antti Valmari, and Tim A. C. Willems</td>
<td></td>
</tr>
<tr>
<td>Semantical Analysis of Contextual Types</td>
<td>502</td>
</tr>
<tr>
<td>Brigitte Pientka and Ulrich Schöpp</td>
<td></td>
</tr>
<tr>
<td>Ambiguity, Weakness, and Regularity in Probabilistic Büchi Automata</td>
<td>522</td>
</tr>
<tr>
<td>Christof Lüding and Anton Pirogov</td>
<td></td>
</tr>
</tbody>
</table>
Local Local Reasoning: A BI-Hyperdoctrine for Full Ground Store .......... 542

Miriam Polzer and Sergey Goncharov

Quantum Programming with Inductive Datatypes: Causality and Affine Type Theory .......................... 562
Romain Péchoux, Simon Perdrix, Mathys Rennela, and Vladimir Zamdzhiev

Spinal Atomic Lambda-Calculus .................................................. 582
David Sherratt, Willem Heijltjes, Tom Gundersen, and Michel Parigot

Learning Weighted Automata over Principal Ideal Domains .................. 602
Gerco van Heerdt, Clemens Kupke, Jurriaan Rot, and Alexandra Silva

The Polynomial Complexity of Vector Addition Systems with States ........ 622
Florian Zuleger

Author Index ................................................................. 643
Neural Flocking: MPC-based Supervised Learning of Flocking Controllers

Usama Mehmoood¹, Shouvik Roy¹, Radu Grosu², Scott A. Smolka¹, Scott D. Stoller¹, and Ashish Tiwari³

¹ Stony Brook University, Stony Brook NY, USA
² Technische Universität Wien, Wien, Austria
³ Microsoft Research, San Francisco CA, USA

Abstract. We show how a symmetric and fully distributed flocking controller can be synthesized using Deep Learning from a centralized flocking controller. Our approach is based on Supervised Learning, with the centralized controller providing the training data, in the form of trajectories of state-action pairs. We use Model Predictive Control (MPC) for the centralized controller, an approach that we have successfully demonstrated on flocking problems. MPC-based flocking controllers are high-performing but also computationally expensive. By learning a symmetric and distributed neural flocking controller from a centralized MPC-based one, we achieve the best of both worlds: the neural controllers have high performance (on par with the MPC controllers) and high efficiency. Our experimental results demonstrate the sophisticated nature of the distributed controllers we learn. In particular, the neural controllers are capable of achieving myriad flocking-oriented control objectives, including flocking formation, collision avoidance, obstacle avoidance, predator avoidance, and target seeking. Moreover, they generalize the behavior seen in the training data to achieve these objectives in a significantly broader range of scenarios. In terms of verification of our neural flocking controller, we use a form of statistical model checking to compute confidence intervals for its convergence rate and time to convergence.

Keywords: Flocking · Model Predictive Control · Distributed Neural Controller · Deep Neural Network · Supervised Learning

1 Introduction

With the introduction of Reynolds rule-based model [16, 17], it is now possible to understand the flocking problem as one of distributed control. Specifically, in this model, at each time-step, each agent executes a control law given in terms of the weighted sum of three competing forces to determine its next acceleration. Each of these forces has its own rule: separation (keep a safe distance away from your neighbors), cohesion (move towards the centroid of your neighbors), and alignment (steer toward the average heading of your neighbors). Reynolds
controller is *distributed*; i.e., it is executed separately by each agent, using information about only itself and nearby agents, and without communication. Furthermore, it is *symmetric*; i.e., every agent runs the same controller (same code).

We subsequently showed that a simpler, more declarative approach to the flocking problem is possible [11]. In this setting, flocking is achieved when the agents combine to minimize a system-wide *cost function*. We presented centralized and distributed solutions for achieving this form of "declarative flocking" (DF), both of which were formulated in terms of Model-Predictive Control (MPC) [2].

Another advantage of DF over the ruled-based approach exemplified by Reynolds model is that it allows one to consider additional control objectives (e.g., obstacle and predator avoidance) simply by extending the cost function with additional terms for these objectives. Moreover, these additional terms are typically quite straightforward in nature. In contrast, deriving behavioral rules that achieve the new control objectives can be a much more challenging task.

An issue with MPC is that computing the next control action can be computationally expensive, as MPC searches for an action sequence that minimizes the cost function over a given prediction horizon. This renders MPC unsuitable for real-time applications with short control periods, for which flocking is a prime example. Another potential problem with MPC-based approaches to flocking is its performance (in terms of achieving the desired flight formation), which may suffer in a fully distributed setting.

In this paper, we present *Neural Flocking* (NF), a new approach to the flocking problem that uses Supervised Learning to learn a symmetric and fully distributed flocking controller from a centralized MPC-based controller. By doing so, we achieve the best of both worlds: high performance (on par with the MPC controllers) in terms of meeting flocking flight-formation objectives, and high efficiency leading to real-time flight controllers. Moreover, our NF controllers can easily be parallelized on hardware accelerators such as GPUs and TPUs.

Figure 1 gives an overview of the NF approach. A high-performing centralized MPC controller provides the labeled training data to the learning agent: a symmetric and distributed neural controller in the form of a deep neural network (DNN). The training data consists of trajectories of state-action pairs, where a state contains the information known to an agent at a time step (e.g., its own position and velocity, and the position and velocity of its neighbors), and the action (the label) is the acceleration assigned to that agent at that time step by the centralized MPC controller.

We formulate and evaluate NF in a number of essential flocking scenarios: basic flocking with inter-agent collision avoidance, as in [11], and more advanced
scenarios with additional objectives, including obstacle avoidance, predator avoidance, and target seeking by the flock. We conduct an extensive performance evaluation of NF. Our experimental results demonstrate the sophisticated nature of NF controllers. In particular, they are capable of achieving all of the stated control objectives. Moreover, they generalize the behavior seen in the training data in order to achieve these objectives in a significantly broader range of scenarios. In terms of verification of our neural controller, we use a form of statistical model checking [5,10] to compute confidence intervals for its rate of convergence to a flock and for its time to convergence.

2 Background

We consider a set of $n$ dynamic agents $\mathcal{A} = \{1, \ldots, n\}$ that move according to the following discrete-time equations of motion:

$$
\begin{align*}
p_i(k+1) &= p_i(k) + dt \cdot v_i(k), \quad |v_i(k)| < \bar{v} \\
v_i(k+1) &= v_i(k) + dt \cdot a_i(k), \quad |a_i(k)| < \bar{a}
\end{align*}
$$

where $p_i(k) \in \mathbb{R}^2$, $v_i(k) \in \mathbb{R}^2$, $a_i(k) \in \mathbb{R}^2$ are the position, velocity and acceleration of agent $i \in \mathcal{A}$ respectively at time step $k$, and $dt \in \mathbb{R}^+$ is the time step. The magnitudes of velocities and accelerations are bounded by $\bar{v}$ and $\bar{a}$, respectively. Acceleration $a_i(k)$ is the control input for agent $i$ at time step $k$. The acceleration is updated after every $\eta$ time steps i.e., $\eta \cdot dt$ is the control period. The flock configuration at time step $k$ is thus given by the following vectors (in boldface):

$$
\begin{align*}
p(k) &= [p_1^T(k) \cdots p_n^T(k)]^T \\
v(k) &= [v_1^T(k) \cdots v_n^T(k)]^T \\
a(k) &= [a_1^T(k) \cdots a_n^T(k)]^T
\end{align*}
$$

The configuration vectors are referred to without the time indexing as $p$, $v$, and $a$. The neighborhood of agent $i$ at time step $k$, denoted by $\mathcal{N}_i(k) \subseteq \mathcal{A}$, contains its $\mathcal{N}$-nearest neighbors, i.e., the $\mathcal{N}$ other agents closest to it. We use this definition (in Section 2.2 to define a distributed-flocking cost function) for simplicity, and expect that a radius-based definition of neighborhood would lead to similar results for our distributed flocking controllers.

2.1 Model-Predictive Control

Model-Predictive control (MPC) [2] is a well-known control technique that has recently been applied to the flocking problem [11,19,20]. At each control step, an optimization problem is solved to find the optimal sequence of control actions (agent accelerations in our case) that minimizes a given cost function with respect to a predictive model of the system. The first control action of the optimal control sequence is then applied to the system; the rest is discarded. In the computation
of the cost function, the predictive model is evaluated for a finite prediction horizon of $T$ control steps.

MPC-based flocking models can be categorized as centralized or distributed. A centralized model assumes that complete information about the flock is available to a single “global” controller, which uses the states of all agents to compute their next optimal accelerations. The following optimization problem is solved by a centralized MPC controller at each control step $k$:

$$\min_{a(k|k), \ldots, a(k+T-1|k) \in \mathcal{A}} J(k) + \lambda \sum_{t=0}^{T-1} \|a(k + t | k)\|^2$$  \hspace{1cm} (5)\)

The first term $J(k)$ is the centralized model-specific cost, evaluated for $T$ control steps (this embodies the predictive aspect of MPC), starting at time step $k$. It encodes the control objective of minimizing the cost function $J(k)$. The second term, scaled by a weight $\lambda > 0$, penalizes large control inputs: $a(k + t | k)$ are the predictions made at time step $k$ for the accelerations at time step $k + t$.

In distributed MPC, each agent computes its acceleration based only on its own state and its local knowledge, e.g., information about its neighbors:

$$\min_{a_i(k|k), \ldots, a_i(k+T-1|k) \in \mathcal{A}} J_i(k) + \lambda \sum_{t=0}^{T-1} \|a_i(k + t | k)\|^2$$ \hspace{1cm} (6)\)

$J_i(k)$ is the distributed, model-specific cost function for agent $i$, analogous to $J(k)$. In a distributed setting where an agent’s knowledge of its neighbors’ behavior is limited, an agent cannot calculate the exact future behavior of its neighbors. Hence, the predictive aspect of $J_i(k)$ must rely on some assumption about that behavior during the prediction horizon. Our distributed cost functions are based on the assumption that the neighbors have zero accelerations during the prediction horizon. While this simple design is clearly not completely accurate, our experiments show that it still achieves good results.

### 2.2 Declarative Flocking

Declarative flocking (DF) is a high-level approach to designing flocking algorithms based on defining a suitable cost function for MPC [11]. This is in contrast to the operational approach, where a set of rules are used to capture flocking behavior, as in Reynolds model. For basic flocking, the DF cost function contains two terms: (1) a cohesion term based on the squared distance between each pair of agents in the flock; and (2) a separation term based on the inverse of the squared distance between each pair of agents. The flock evolves toward a configuration in which these two opposing forces are balanced. The cost function $J^C$ for centralized DF, i.e., centralized MPC (CMPC), is as follows:

$$J^C(p) = \frac{2}{|\mathcal{A}| \cdot (|\mathcal{A}| - 1)} \sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{A}, j < i} \|p_{ij}\|^2 + \omega_s \cdot \frac{1}{\|p_{ij}\|^2}$$ \hspace{1cm} (7)\)
where $\omega_s$ is the weight of the separation term and controls the density of the flock. The cost function is normalized by the number of pairs of agents, $|\mathcal{A}|(\mathcal{A}-1)/2$; as such, the cost does not depend on the size of the flock. The control law for CMPC is given by Eq. (5), with $J(k) = \sum_{i=1}^{T} J_i^C(p(k + t | k))$.

The basic flocking cost function for distributed DF is similar to that for CMPC, except that the cost function $J_i^D$ for agent $i$ is computed over its set of neighbors $\mathcal{N}_i(k)$ at time $k$:

$$J_i^D(p(k)) = \frac{1}{|\mathcal{N}_i(k)|} \cdot \sum_{j \in \mathcal{N}_i(k)} \|p_{ij}\|^2 + \omega_s \cdot \sum_{j \in \mathcal{N}_i(k)} \frac{1}{\|p_{ij}\|^2} \quad (8)$$

The control law for agent $i$ is given by Eq. (6), with $J_i(k) = \sum_{t=1}^{T} J_i^D(p(k + t | k))$.

3 Additional Control Objectives

The cost functions for basic flocking given in Eqs. (7) and (8) are designed to ensure that in the steady state, the agents are well-separated. Additional goals such as obstacle avoidance, predator avoidance, and target seeking are added to the MPC formulation as weighted cost-function terms. Different objectives can be combined by including the corresponding terms in the cost function as a weighted sum.

Cost-Function Term for Obstacle Avoidance. We consider multiple rectangular obstacles which are distributed randomly in the field. For a set of $m$ rectangular obstacles $\mathcal{O} = \{O_1, O_2, ..., O_m\}$, we define the cost function term for obstacle avoidance as:

$$J_{OA}(p, o) = \frac{1}{|\mathcal{A}| |\mathcal{O}|} \sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{O}} \frac{1}{\|p_i - o_j^{(i)}\|^2} \quad (9)$$

where $o$ is the set of points on the obstacle boundaries and $o_j^{(i)}$ is the point on the obstacle boundary of the $j^{th}$ obstacle $O_j$ that is closest to the $i^{th}$ agent.

Cost-Function Term for Target Seeking. This term is the average of the squared distance between the agents and the target. Let $g$ denote the position of the fixed target. Then the target-seeking term is as defined as

$$J_{TS}(p) = \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} \|p_i - g\|^2 \quad (10)$$

Cost-Function Term for Predator Avoidance. We introduce a single predator, which is more agile than the flocking agents: its maximum speed and acceleration are a factor of $f_p$ greater than $\bar{v}$ and $\bar{a}$, respectively, with $f_p > 1$. Apart from being more agile, the predator has the same dynamics as the agents, given by
Eq. (1). The control law for the predator consists of a single term that causes it to move toward the centroid of the flock with maximum acceleration.

For a flock of \( n \) agents and one predator, the cost-function term for predator avoidance is the average of the inverse of the cube of the distances between the predator and the agents. It is given by:

\[
J_{PA}(p, p_{pred}) = \frac{1}{|A|} \sum_{i \in A} \frac{1}{\|p_i - p_{pred}\|^3}
\]

(11)

where \( p_{pred} \) is the position of the predator. In contrast to the separation term in Eqs. (5)-(6), which we designed to ensure inter-agent collision avoidance, the predator-avoidance term has a cube instead of a square in the denominator. This is to reduce the influence of the predator on the flock when the predator is far away from the flock.

**NF Cost-Function Terms.** The MPC cost functions used in our examination of Neural Flocking are weighted sums of the cost function terms introduced above. We refer to the first term of our centralized DF cost function \( J^C(p) \) (see Eq. (7)) as \( J_{coh}(p) \) and the second as \( J_{sep}(p) \). We use the following cost functions \( J_1 \), \( J_2 \), and \( J_3 \) for basic flocking with collision avoidance, obstacle avoidance with target seeking, and predator avoidance, respectively.

\[
J_1(p) = J_{coh}(p) + \omega_s \cdot J_{sep}(p)
\]

(12a)

\[
J_2(p, o) = J_{coh}(p) + \omega_s \cdot J_{sep}(p) + \omega_o \cdot J_{OA}(p, o) + \omega_t \cdot J_{TS}(p)
\]

(12b)

\[
J_3(p, p_{pred}) = J_{coh}(p) + \omega_s \cdot J_{sep}(p) + \omega_p \cdot J_{PA}(p, p_{pred})
\]

(12c)

where \( \omega_s \) is the weight of the separation term, \( \omega_o \) is the weight of the obstacle avoidance term, \( \omega_t \) is the weight of the target-seeking term, and \( \omega_p \) is the weight of the predator-avoidance term. Note that \( J_1 \) is equivalent to \( J^C \) (Eq. (7)). The weight \( \omega_s \) of the separation term is experimentally chosen to ensure that the distance between agents, throughout the simulation, is at least \( d_{min} \), the minimum inter-agent distance representing collision avoidance. Similar considerations were given to the choice of values for \( \omega_o \) and \( \omega_p \). The specific values we used for the weights are: \( \omega_s = 2000, \omega_o = 1500, \omega_t = 10, \) and \( \omega_p = 500. \)

We experimented with an alternative strategy for introducing inter-agent collision avoidance, obstacle avoidance, and predator avoidance into the MPC problem, namely, as constraints of the form \( d_{min} - p_{ij} < 0, \ d_{min} - \|p_i - o_j\| < 0, \) and \( d_{min} - \|p_i - p_{pred}\| < 0 \), respectively. Using the theory of exact penalty functions [12], we recast the constrained MPC problem as an equivalent unconstrained MPC problem by converting the constraints into a weighted penalty term, which is then added to the MPC cost function. This approach rendered the optimization problem difficult to solve due to the non-smoothness of the penalty term. As a result, constraint violations in the form of collisions were observed during simulation.
4 Neural Flocking

We learn a distributed neural controller (DNC) for the flocking problem using training data in the form of trajectories of state-action pairs produced by a CMPC controller. In addition to basic flocking with inter-agent collision avoidance, the DNC exhibits a number of other flocking-related behaviors, including obstacle avoidance, target seeking, and predator avoidance. We also show how the learned behavior exhibited by the DNC generalizes over a larger number of agents than what was used during training to achieve successful collision-free flocking in significantly larger flocks.

We use Supervised Learning to train the DNC. Supervised Learning learns a function that maps an input to an output based on example sequences of input-output pairs. In our case, the trajectory data obtained from CMPC contains both the training inputs and corresponding labels (outputs): the state of an agent in the flock (and that of its nearest neighbors) at a particular time step is the input, and that agent’s acceleration at the same time step is the label.

4.1 Training Distributed Flocking Controllers

We use Deep Learning to synthesize a distributed and symmetric neural controller from the training data provided by the CMPC controller. Our objective is to learn basic flocking, obstacle avoidance with target seeking, and predator avoidance. Their respective CMPC-based cost functions are given in Sections 2.2 and 3. All of these control objectives implicitly also include inter-agent collision avoidance by virtue of the separation term in Eq. 7.

For each of these control objectives, DNC training data is obtained from CMPC trajectory data generated for \( n = 15 \) agents, starting from initial configurations in which agent positions and velocities are uniformly sampled from \([-15, 15]^2\) and \([0, 1]^2\), respectively. All training trajectories are 1,000 time steps in duration.

We further ensure that the initial configurations are recoverable; i.e., no two agents are so close to each other that they cannot avoid a collision by resorting to maximal accelerations. We learn a single DNC from the state-action pairs of all \( n \) agents. This yields a symmetric distributed controller, which we use for each agent in the flock during evaluation.

Basic Flocking. Trajectory data for basic flocking is generated using the cost function given in Eq. (7). We generate 200 trajectories, each of which (as noted above) is 1,000 time steps long. The input to the NN is the position and velocity of each agent along with the positions and velocities of its \( \mathcal{N} \)-nearest neighbors. This yields 200 \( \cdot \) 1,000 \( \cdot \) 15 = 3M total training samples.

Let us refer to the agent (the DNC) being learned as \( \mathcal{A}_0 \). Since we use neighborhood size \( \mathcal{N} = 14 \), the input to the NN is of the form \( [p_0^x \ p_0^y \ v_0^x \ v_0^y \ p_1^x \ p_1^y \ v_1^x \ v_1^y \ \ldots \ p_{14}^x \ p_{14}^y \ v_{14}^x \ v_{14}^y] \), where \( p_0^x, p_0^y \) are the position coordinates and \( v_0^x, v_0^y \) velocity coordinates for agent \( \mathcal{A}_0 \), and \( p_1^{x\ldots 14}, p_1^{y\ldots 14} \) and \( v_1^{x\ldots 14}, v_1^{y\ldots 14} \) are the position and velocity vectors of its neighbors. Since this input vector has 60 components, the input to the NN consists of 60 features.
Obstacle Avoidance with Target Seeking. For obstacle avoidance with target seeking, we use CMPC with the cost function given in Eq. (12b). The target is located beyond the obstacles, forcing the agents to move through the obstacle field. For the training data, we generate 100 trajectories over 4 different obstacle fields (25 trajectories per obstacle field). The input to the NN consists of the 92 features $[p_0^x p_0^y v_0^x v_0^y o_0^x o_0^y \ldots p_{14}^x p_{14}^y v_{14}^x v_{14}^y o_{14}^x o_{14}^y g_x^v g_y^v]$, where $o_k^x$, $o_k^y$ is the closest point on any obstacle to agent $A_0$; $o_{14}^x$, $o_{14}^y$ give the closest point on any obstacle for the 14 neighboring agents, and $g_x^v$, $g_y^v$ is the target location.

Predator Avoidance. The CMPC cost function for predator avoidance is given in Eq. (12c). The position, velocity, and the acceleration of the predator are denoted by $p_{pred}$, $v_{pred}$, $a_{pred}$, respectively. We take $f_p = 1.40$; hence $\dot{v}_{pred} = 1.40 \bar{v}$ and $\ddot{a}_{pred} = 1.40 \bar{a}$. The input features to the NN are the positions and velocities of agent $A_0$ and its $N$-nearest neighbors, and the position and velocity of the predator. The input with 64 features thus has the form $[p_0^x p_0^y v_0^x v_0^y \ldots p_{14}^x p_{14}^y v_{14}^x v_{14}^y p_{pred}^x p_{pred}^y v_{pred}^x v_{pred}^y]$.

5 Experimental Evaluation

This section contains the results of our extensive performance analysis of the distributed neural flocking controller (DNC), taking into account various control objectives: basic flocking with collision avoidance, obstacle avoidance with target seeking, and predator avoidance. As illustrated in Fig. 1, this involves running CMPC to generate the training data for the DNCs, whose performance we then compare to that of the DMPC and CMPC controllers. We also show that the DNC flocking controllers generalize the behavior seen in the training data to achieve successful collision-free flocking in flocks significantly larger in size than those used during training. Finally, we use Statistical Model Checking to obtain confidence intervals for DNC’s correctness/performance.

5.1 Preliminaries

The CMPC and DMPC control problems defined in Section 2.1 are solved using MATLAB fmincon optimizer. In the training phase, the size of the flock is
$n = 15$. For obstacle-avoidance with target-seeking, we use 5 obstacles with the target located at $[60,50]$. The simulation time is 100, $dt = 0.1$ time units, and $\eta = 3$, where (recall) $\eta \cdot dt$ is the control period. Further, the agent velocity and acceleration bounds are $\bar{v} = 2.0$ and $\bar{a} = 1.5$.

We use $d_{\text{min}} = 1.5$ as the minimum inter-agent distance for collision avoidance, $d_{\text{obs}}^\text{min} = 1$ as the minimum agent-obstacle distance for obstacle avoidance, and $d_{\text{pred}}^\text{min} = 1.5$ as the minimum agent-predator distance for predator avoidance. For initial configurations, recall that agent positions and velocities are uniformly sampled from $[-15,15]^2$ and $[0,1]^2$, respectively, and we ensure that they are recoverable; i.e., no two agents are so close to each other that they cannot avoid a collision when resorting to maximal accelerations. The predator starts at rest from a fixed location at a distance of 40 from the flock center.

For training, we considered 15 agents and 200 trajectories per agent, each trajectory 1,000 time steps in length. This yielded a total of 3,000,000 training samples. Our neural controller is a fully connected feed-forward Deep Neural Network (DNN), with 5 hidden layers, 84 neurons per hidden layer, and with a ReLU activation function. We use an iterative approach for choosing the DNN hyperparameters and architecture where we continuously improve our NN, until we observe satisfactory performance by the DNC.

For training the DNNs, we use Keras [3], which is a high-level neural network API written in Python and capable of running on top of TensorFlow. To generate the NN model, Keras uses the Adam optimizer [8] with the following settings: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. The batch size (number of samples processed before the model is updated) is 2,000, and the number of epochs (number of complete passes through the training dataset) used for training is 1,000. For measuring training loss, we use the mean-squared error metric.

For basic flocking, DNN input vectors have 60 features and the number of trainable DNN parameters is 33,854. For flocking with obstacle-avoidance and target-seeking, input vectors have 92 features and the number of trainable parameters is 36,542. Finally, for flocking with predator-avoidance, input vectors have 64 features and the resulting number of trainable DNN parameters is 34,190.

To test the trained DNC, we generated 100 simulations (runs) for each of the desired control objectives: basic flocking with collision avoidance, flocking with obstacle avoidance and target seeking, and flocking with predator avoidance. The results presented in Tables 1, were obtained using the same number of agents and obstacles and the same predator as in the training phase. We also ran tests that show DNC controllers can achieve collision-free flocking with obstacle avoidance where the numbers of agents and obstacles are greater than those used during training.

### 5.2 Results for Basic Flocking

We use flock diameter, inter-agent collision count and velocity convergence [20] as performance metrics for flocking behavior. At any time step, the flock diameter $D(p) = \max_{(i,j) \in A} \| p_{ij} \|$ is the largest distance between any two agents in the flock. We calculate the average converged diameter by averaging the flock diameter
in the final time step of the simulation over the 100 runs. An inter-agent collision (IC) occurs when the distance between two agents at any point in time is less than $a_{\text{min}}$. The IC rate (ICR) is the average number of ICs per test-trajectory timestep. The velocity convergence $VC(v) = (1/n) \left( \sum_{i \in A} ||v_i - (\sum_{j=1}^{n} v_j/n)||^2 \right)$ is the average of the squared magnitude of the discrepancy between the velocities of agents and the flock’s average velocity. For all the metrics, lower values are better, indicating a denser and more coherent flock with fewer collisions. A successful flocking controller should also ensure that values of $D(p)$ and $VC(v)$ eventually stabilize.

Fig. 3 and Table 1 compare the performance of the DNC on the basic-flocking problem for 15 agents to that of the MPC controllers. Although the DMPC and CMPC outperform the DNC, the difference is marginal. An important advantage of the DNC over DMPC is that they are much faster. Executing a DNC controller requires a modest number of arithmetic operations, whereas executing an MPC controller requires simulation of a model and controller over the prediction horizon. In our experiments, on average, the CMPC takes 1209 msec of CPU time for the entire flock and DMPC takes 58 msec of CPU time per agent, whereas the DNC takes only 1.6 msec.

<table>
<thead>
<tr>
<th></th>
<th>Avg. Conv. Diameter</th>
<th>ICR</th>
<th>Velocity Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNC</td>
<td>14.13</td>
<td>0</td>
<td>0.15</td>
</tr>
<tr>
<td>DMPC</td>
<td>13.67</td>
<td>0</td>
<td>0.11</td>
</tr>
<tr>
<td>CMPC</td>
<td>13.84</td>
<td>0</td>
<td>0.10</td>
</tr>
</tbody>
</table>
Table 2: DNC Performance Generalization for BF

<table>
<thead>
<tr>
<th>Agents</th>
<th>Avg. Conv. Diameter</th>
<th>Conv. Rate (%)</th>
<th>Avg. Conv. Time</th>
<th>ICR</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14.13</td>
<td>100</td>
<td>52.15</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>16.45</td>
<td>97</td>
<td>58.76</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>19.81</td>
<td>94</td>
<td>64.11</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>23.24</td>
<td>92</td>
<td>72.08</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>30.57</td>
<td>86</td>
<td>83.84</td>
<td>0.008</td>
</tr>
<tr>
<td>40</td>
<td>38.66</td>
<td>81</td>
<td>95.32</td>
<td>0.019</td>
</tr>
</tbody>
</table>

5.3 Results for Obstacle and Predator Avoidance

For obstacle and predator avoidance, collision rates are used as a performance metric. An obstacle-agent collision (OC) occurs when the distance between an agent and the closest point on any obstacle is less than \( d_{\text{min}}^{\text{obs}} \). A predator-agent collision (PC) occurs when the distance between an agent and the predator is less than \( d_{\text{min}}^{\text{pred}} \). The OC rate (OCR) is the average number of OCs per test-trajectory time-step, and the PC rate (PCR) is defined similarly. Our test results show that the DNC, along with the DMPC and CMPC, is collision-free (i.e., each of ICR, OCR, and PCR is zero) for 15 agents, with the exception of DMPC for predator avoidance where PCR = 0.013. We also observed that the flock successfully reaches the target location in all 100 test runs.

5.4 DNC Generalization Results

Tables 2–3 present DNC generalization results for basic flocking (BF), obstacle avoidance (OA), and predator avoidance (PA), with the number of agents ranging from 15 (the flock size during training) to 40. In all of these experiments, we use a neighborhood size of \( N = 14 \), the same as during training. Each controller was evaluated with 100 test runs. The performance metrics in Table 2 are the average converged diameter, convergence rate, average convergence time, and ICR.

The convergence rate is the fraction of successful flocks over 100 runs. The collection of agents is said to have converged to a flock (with collision avoidance) if the value of the global cost function is less than the convergence threshold. We use a convergence threshold of \( J_1(p) \leq 150 \), which was chosen based on its proximity to the value achieved by CMPC. We use the cost function from Eq. 12a to calculate our success rate because we are showing convergence rate for basic flocking. The average convergence time is the time when the global cost function first drops below the success threshold and remains below it for the rest of the run, averaged over all 100 runs. Even with a local neighborhood of size 14, the results demonstrate that the DNC can successfully generalize to a large number of agents for all of our control objectives.
5.5 Statistical Model Checking Results

We use Monte Carlo (MC) approximation as a form of Statistical Model Checking [5, 10] to compute confidence intervals for the DNC’s convergence rate to a flock with collision avoidance and for the (normalized) convergence time. The convergence rate is the fraction of successful flocks over N runs. The collection of agents is said to have converged to a successful flock with collision avoidance if the global cost function $J_1(p) \leq 150$, where $J_1(p)$ is cost function for basic flocking defined in Eq. 12a.

The main idea of MC is to use $N$ random variables, $Z_1, \ldots, Z_N$, also called samples, IID distributed according to a random variable $Z$ with mean $\mu_Z$, and to take the sum $\bar{\mu}_Z = (Z_1 + \ldots + Z_N)/N$ as the value approximating the mean $\mu_Z$. Since an exact computation of $\mu_Z$ is almost always intractable, an MC approach is used to compute an $(\epsilon, \delta)$-approximation of this quantity.

Additive Approximation [6] is an $(\epsilon, \delta)$-approximation scheme where the mean $\mu_Z$ of an RV $Z$ is approximated with absolute error $\epsilon$ and probability $1 - \delta$:

$$Pr[\mu_Z - \epsilon \leq \bar{\mu}_Z \leq \mu_Z + \epsilon] \geq 1 - \delta$$

(13)

where $\bar{\mu}_Z$ is an approximation of $\mu_Z$. An important issue is to determine the number of samples $N$ needed to ensure that $\bar{\mu}_Z$ is an $(\epsilon, \delta)$-approximation of $\mu_Z$. If $Z$ is a Bernoulli variable expected to be large, one can use the Chernoff-Hoeffding instantiation of the Bernstein inequality and take $N$ to be $N = 4 \ln(2/\delta)/\epsilon^2$, as in [6]. This results in the additive approximation algorithm [5], defined in Algorithm 1.

We use this algorithm to obtain a joint $(\epsilon, \delta)$-approximation of the mean convergence rate and mean normalized convergence time for the DNC. Each sample $Z_i$ is based on the result of an execution obtained by simulating the system starting from a random initial state, and we take $Z = (B, R)$, where $B$ is a Boolean variable indicating whether the agents converged to a flock during the execution, and $R$ is a real value denoting the normalized convergence time. The normalized convergence time is the time when the global cost function first drops below the convergence threshold and remains below it for the rest of the run, measured as a fraction of the total duration of the run. The assumptions
Algorithm 1: Additive Approximation Algorithm

Input: \((\epsilon, \delta)\) with \(0 < \epsilon < 1\) and \(0 < \delta < 1\)
Input: Random variables \(Z_i\), IID
Output: \(\hat{\mu}_Z\) approximation of \(\mu_Z\)
\(N = 4\ln(2/\delta)/\epsilon^2;\)
for (\(i=0; i < N; i++\)) do
\(S = S + Z_i;\)
\(\hat{\mu}_Z = S/N;\) return \(\hat{\mu}_Z;\)

Table 4: SMC results for DNC convergence rate and normalized convergence time; \(\epsilon = 0.01, \delta = 0.0001\)

<table>
<thead>
<tr>
<th>Agents</th>
<th>(\hat{\mu}_{CR})</th>
<th>(\hat{\mu}_{CT})</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.99</td>
<td>0.53</td>
</tr>
<tr>
<td>20</td>
<td>0.97</td>
<td>0.58</td>
</tr>
<tr>
<td>25</td>
<td>0.94</td>
<td>0.65</td>
</tr>
<tr>
<td>30</td>
<td>0.91</td>
<td>0.71</td>
</tr>
<tr>
<td>35</td>
<td>0.86</td>
<td>0.84</td>
</tr>
<tr>
<td>40</td>
<td>0.80</td>
<td>0.95</td>
</tr>
</tbody>
</table>

about \(Z\) required for validity of the additive approximation hold, because RV \(B\)
is a Bernoulli variable, the convergence rate is expected to be large (i.e., closer
to 1 than to 0), and the proportionality constraint of the Bernstein inequality is
also satisfied for RV \(R\).

In these experiments, the initial configurations are sampled from the same
distributions as in Section 5.1, and we set \(\epsilon = 0.01\) and \(\delta = 0.0001\), to obtain \(N = 396,140\). We perform the required set of \(N\) simulations for 15, 20, 25, 30, 35 and
40 agents. Table 4 presents the results, specifically, the \((\epsilon, \delta)\)-approximations \(\hat{\mu}_{CR}\)
and \(\hat{\mu}_{CT}\) of the mean convergence rate and the mean normalized convergence
time, respectively. While the results for the convergence rate are (as expected) nu-
merically similar to the results in Table 2, the results in Table 4 are much stronger,
because they come with the guarantee that they are \((\epsilon, \delta)\)-approximations of the
actual mean values.

6 Related Work

In [18], a flocking controller is synthesized using multi-agent reinforcement learning
(MARL) and natural evolution strategies (NES). The target model from which
the system learns is Reynolds’ flocking model [16]. For training purposes, a list
of metrics called entropy are chosen, which provide a measure of the collective
behavior displayed by the target model. As the authors of [18] observe, this
technique does not quite work: although it consistently leads to agents forming
recognizable patterns during simulation, agents self-organized into a cluster
instead of flowing like a flock.
In [9], reinforcement learning and flocking control are combined for the purpose of predator avoidance, where the learning module determines safe spaces in which the flock can navigate to avoid predators. Their approach to predator avoidance, however, isn’t distributed as it requires a majority consensus by the flock to determine its action to avoid predators. They also impose an $\alpha$-lattice structure [13] on the flock. In contrast, our approach is geometry-agnostic and achieves predator avoidance in a distributed manner.

In [7], an uncertainty-aware reinforcement learning algorithm is developed to estimate the probability of a mobile robot colliding with an obstacle in an unknown environment. Their approach is based on bootstrap neural networks using dropouts, allowing it to process raw sensory inputs. Similarly, a learning-based approach to robot navigation and obstacle avoidance is presented in [14]. They train a model that maps sensor inputs and the target position to motion commands generated by the ROS [15] navigation package. Our work in contrast considers obstacle avoidance (and other control objectives) in a multi-agent flocking scenario under the simplifying assumption of full state observation.

In [4], an approach based on Bayesian inference is proposed that allows an agent in a heterogeneous multi-agent environment to estimate the navigation model and goal of each of its neighbors. It then uses this information to compute a plan that minimizes inter-agent collisions while allowing the agent to reach its goal. Flocking formation is not considered.

7 Conclusions

With the introduction of Neural Flocking (NF), we have shown how machine learning in the form of Supervised Learning can bring many benefits to the flocking problem. As our experimental evaluation confirms, the symmetric and fully distributed neural controllers we derive in this manner are capable of achieving a multitude of flocking-oriented objectives, including flocking formation, inter-agent collision avoidance, obstacle avoidance, predator avoidance, and target seeking. Moreover, NF controllers exhibit real-time performance and generalize the behavior seen in the training data to achieve these objectives in a significantly broader range of scenarios.

Ongoing work aims to determine whether a DNC can perform as well as the centralized MPC controller for agent models that are significantly more realistic than our current point-based model. For this purpose, we are using transfer learning to train a DNC that can achieve acceptable performance on realistic quadrotor dynamics [1], starting from our current point-model-based DNC. This effort also involves extending our current DNC from 2-dimensional to 3-dimensional spatial coordinates. If successful, and preliminary results are encouraging, this line of research will demonstrate that DNCs are capable of achieving flocking with complex realistic dynamics.

For future work, we plan to investigate a distance-based notion of agent neighborhood as opposed to our current nearest-neighbors formulation. Furthermore, motivated by the quadrotor study of [21], we will seek to combine MPC with
reinforcement learning in the framework of guided policy search as an alternative solution technique for the NF problem.

References


Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.
On Well-Founded and Recursive Coalgebras

Jiří Adámek1,**, Stefan Milius2,***, and Lawrence S. Moss3,†

1 Czech Technical University, Prague, Czech Republic
   j.adamek@tu-braunschweig.de
2 Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
   mail@stefan-milius.eu
3 Indiana University, Bloomington, IN, USA
   lmoss@indiana.edu

Abstract This paper studies fundamental questions concerning category-theoretic models of induction and recursion. We are concerned with the relationship between well-founded and recursive coalgebras for an endofunctor. For monomorphism preserving endofunctors on complete and well-powered categories every coalgebra has a well-founded part, and we provide a new, shorter proof that this is the coreflection in the category of all well-founded coalgebras. We present a new more general proof of Taylor’s General Recursion Theorem that every well-founded coalgebra is recursive, and we study conditions which imply the converse. In addition, we present a new equivalent characterization of well-foundedness: a coalgebra is well-founded iff it admits a coalgebra-to-algebra morphism to the initial algebra.

Keywords: Well-founded · Recursive · Coalgebra · Initial Algebra · General Recursion Theorem

1 Introduction

What is induction? What is recursion? In areas of theoretical computer science, the most common answers are related to initial algebras. Indeed, the dominant trend in abstract data types is initial algebra semantics (see e.g. [19]), and this approach has spread to other semantically-inclined areas of the subject. The approach in broad slogans is that, for an endofunctor $F$ describing the type of algebraic operations of interest, the initial algebra $\mu F$ has the property that for every $F$-algebra $A$, there is a unique homomorphism $\mu F \to A$, and this is recursion. Perhaps the primary example is recursion on $\mathbb{N}$, the natural numbers. Recall that $\mathbb{N}$ is the initial algebra for the set functor $FX = X + 1$. If $A$ is any set, and $a \in A$ and $\alpha : A \to A + 1$ are given, then initiality tells us that there is a unique $f : \mathbb{N} \to A$ such that for all $n \in \mathbb{N}$,

$$f(0) = a \quad f(n + 1) = \alpha(f(n)).$$

(1.1)

* A full version of this paper including full proof details is available on arXiv [5].
** Supported by the Grant Agency of the Czech Republic under grant 19-00902S.
*** Supported by Deutsche Forschungsgemeinschaft (DFG) under project MI 717/5-2.
† Supported by grant #586136 from the Simons Foundation.

© The Author(s) 2020
https://doi.org/10.1007/978-3-030-45231-5_2
Then the first additional problem coming with this approach is that of how to “recognize” initial algebras: Given an algebra, how do we really know if it is initial? The answer — again in slogans — is that initial algebras are the ones with “no junk and no confusion.”

Although initiality captures some important aspects of recursion, it cannot be a fully satisfactory approach. One big missing piece concerns recursive definitions based on well-founded relations. For example, the whole study of termination of rewriting systems depends on well-orders, the primary example of recursion on a well-founded order. Let $(X, R)$ be a well-founded relation, i.e. one with no infinite sequences \( \cdots x_2 R x_1 R x_0 \). Let $A$ be any set, and let $\alpha: \mathcal{P}A \to A$. (Here and below, $\mathcal{P}$ is the power set functor, taking a set to the set of its subsets.)

Then there is a unique $f: X \to A$ such that for all $x \in X$,

$$f(x) = \alpha(\{ f(y) : y R x \}).$$

(1.2)

The main goal of this paper is the study of concepts that allow one to extend the algebraic spirit behind initiality in (1.1) to the setting of recursion arising from well-foundedness as we find it in (1.2). The corresponding concepts are those of well-founded and recursive coalgebras for an endofunctor, which first appear in work by Osius [22] and Taylor [23, 24], respectively. In his work on categorical set theory, Osius [22] first studied the notions of well-founded and recursive coalgebras (for the power-set functor on sets and, more generally, the power-object functor on an elementary topos). He defined recursive coalgebras as those coalgebras $\alpha: A \to \mathcal{P}A$ which have a unique coalgebra-to-algebra homomorphism into every algebra (see Definition 3.2).

Taylor [23, 24] took Osius’ ideas much further. He introduced well-founded coalgebras for a general endofunctor, capturing the notion of a well-founded relation categorically, and considered recursive coalgebras under the name ‘coalgebras obeying the recursion scheme’. He then proved the General Recursion Theorem that all well-founded coalgebras are recursive, for every endofunctor on sets (and on more general categories) preserving inverse images. Recursive coalgebras were also investigated by Eppendahl [12], who called them algebra-initial coalgebras. Capretta, Uustalu, and Vene [10] further studied recursive coalgebras, and they showed how to construct new ones from given ones by using comonads. They also explained nicely how recursive coalgebras allow for the semantic treatment of (functional) divide-and-conquer programs. More recently, Jeannin et al. [15] proved the General Recursion Theorem for polynomial functors on the category of many-sorted sets; they also provide many interesting examples of recursive coalgebras arising in programming.

Our contributions in this paper are as follows. We start by recalling some preliminaries in Section 2 and the definition of (parametrically) recursive coalgebras in Section 3 and of well-founded coalgebras in Section 4 (using a formulation based on Jacobs’ next time operator [14], which we extend from Kripke polynomial set functors to arbitrary functors). We show that every coalgebra for a monomorphism preserving functor on a complete and well-powered category has a well-founded part, and provide a new proof that this is the coreflection in the
category of well-founded coalgebras (Proposition 4.19), shortening our previous
proof [6]. Next we provide a new proof of Taylor’s General Recursion Theorem
(Theorem 5.1), generalizing this to endofunctors preserving monomorphisms on a
complete and well-powered category having smooth monomorphisms (see Defini-
tion 2.8). For the category of sets, this implies that “well-founded ⇒ recursive”
holds for all endofunctors, strengthening Taylor’s result. We then discuss the
converse: is every recursive coalgebra well-founded? Here the assumption that \( F \)
preserves inverse images cannot be lifted, and one needs additional assumptions.
In fact, we present two results: one assumes universally smooth monomorph-
isms and that the functor has a pre-fixed point (see Theorem 5.5). Under these
assumptions we also give a new equivalent characterization of recursiveness
and well-foundedness: a coalgebra is recursive if it has a coalgebra-to-algebra
morphism into the initial algebra (which exists under our assumptions), see Cor-
rollary 5.6. This characterization was previously established for finitary functors
on sets [3]. The other converse of the above implication is due to Taylor using
the concept of a subobject classifier (Theorem 5.8). It implies that ‘recursive’
and ‘well-founded’ are equivalent concepts for all set functors preserving inverse
images. We also prove that a similar result holds for the category of vector spaces
over a fixed field (Theorem 5.12).

Finally, we show in Section 6 that well-founded coalgebras are closed under
coproducts, quotients and, assuming mild assumptions, under subcoalgebras.

2 Preliminaries

We start by recalling some background material. Except for the definitions of
algebra and coalgebra in Subsection 2.1, the subsections below may be read as
needed. We assume that readers are familiar with notions of basic category theory;
see e.g. [2] for everything which we do not detail. We indicate monomorphisms
by writing \( \rightarrow \) and strong epimorphisms by \( \twoheadrightarrow \).

2.1 Algebras and Coalgebras. We are concerned throughout this paper
with algebras and coalgebras for an endofunctor. This means that we have an
underlying category, usually written \( \mathcal{A} \); frequently it is the category of sets or
of vector spaces over a fixed field, and that a functor \( F: \mathcal{A} \to \mathcal{A} \) is given. An
\( F \)-algebra is a pair \( (A, \alpha) \), where \( \alpha: FA \to A \). An \( F \)-coalgebra is a pair \( (A, \alpha) \),
where \( \alpha: A \to FA \). We usually drop the functor \( F \). Given two algebras \( (A, \alpha) \)
and \( (B, \beta) \), an algebra homomorphism from the first to the second is \( h: A \to B \n\) in \( \mathcal{A} \) such that \( h \cdot \alpha = \beta \cdot Fh \). Similarly, a coalgebra homomorphism satisfies
\( \beta \cdot h = Fh \cdot \alpha \). We denote by \( \text{Coalg} \ F \) the category of all coalgebras for \( F \).

Example 2.1. (1) The power set functor \( \mathcal{P} : \text{Set} \to \text{Set} \) takes a set \( X \) to the set
\( \mathcal{P}X \) of all subsets of it; for a morphism \( f: X \to Y \), \( \mathcal{P}f : \mathcal{P}X \to \mathcal{P}Y \) takes a
subset \( S \subseteq X \) to its direct image \( f[S] \). Coalgebras \( \alpha: X \to \mathcal{P}X \) may be identified
with directed graphs on the set \( X \) of vertices, and the coalgebra structure \( \alpha \)
describes the edges: \( b \in \alpha(a) \) means that there is an edge \( a \to b \) in the graph.
(2) Let $\Sigma$ be a signature, i.e. a set of operation symbols, each with a finite arity. The polynomial functor $H_\Sigma$ associated to $\Sigma$ assigns to a set $X$ the set

$$H_\Sigma X = \bigoplus_{n \in \mathbb{N}} \Sigma_n \times X^n,$$

where $\Sigma_n$ is the set of operation symbols of arity $n$. This may be identified with the set of all terms $\sigma(x_1, \ldots, x_n)$, for $\sigma \in \Sigma_n$, and $x_1, \ldots, x_n \in X$. Algebras for $H_\Sigma$ are the usual $\Sigma$-algebras.

(3) Deterministic automata over an input alphabet $\Sigma$ are coalgebras for the functor $FX = \{0,1\} \times X^\Sigma$. Indeed, given a set $S$ of states, a next-state map $S \times \Sigma \to S$ may be curried to $\delta : S \to S^\Sigma$. The set of final states yields the acceptance predicate $a : S \to \{0,1\}$. So an automaton may be regarded as a coalgebra $(a, \delta) : S \to \{0,1\} \times S^\Sigma$.

(4) Labelled transitions systems are coalgebras for $FX = \mathcal{P}(\Sigma \times X)$.

(5) To describe linear weighted automata, i.e. weighted automata over the input alphabet $\Sigma$ with weights in a field $K$, as coalgebras, one works with the category $\text{Vec}_K$ of vector spaces over $K$. A linear weighted automaton is then a coalgebra for $FX = K \times X^\Sigma$.

### 2.2 Preservation Properties.

Recall that an intersection of two subobjects $s_i : S_i \hookrightarrow A$ ($i = 1, 2$) of a given object $A$ is given by their pullback. Analogously, (general) intersections are given by wide pullbacks. Furthermore, the inverse image of a subobject $s : S \to B$ under a morphism $f : A \to B$ is the subobject $t : T \to A$ obtained by a pullback of $s$ along $f$.

All of the ‘usual’ set functors preserve intersections and inverse images:

**Example 2.2.**

1. Every polynomial functor preserves intersections and inverse images.
2. The power-set functor $\mathcal{P}$ preserves intersections and inverse images.
3. Intersection-preserving set functors are closed under taking coproducts, products and composition. Similarly, for inverse images.
4. Consider next the set functor $R$ defined by $RX = \{(x, y) \in X \times X : x \neq y\} + \{d\}$ for sets $X$. For a function $f : X \to Y$ put $Rf(x, y) = (f(x), f(y))$ if $f(x) \neq f(y)$, and $d$ otherwise. $R$ preserves intersections but not inverse images.

**Proposition 2.3** [27]. For every set functor $F$ there exists an essentially unique set functor $\tilde{F}$ which coincides with $F$ on nonempty sets and functions and preserves finite intersections (whence monomorphisms).

**Remark 2.4.**

1. In fact, Trnková gave a construction of $\tilde{F}$: she defined $\tilde{F}\emptyset$ as the set of all natural transformations $C_{01} \to F$, where $C_{01}$ is the set functor with $C_{01}\emptyset = \emptyset$ and $C_{01}X = 1$ for all nonempty sets $X$. For the empty map $e : \emptyset \to X$ with $X \neq \emptyset$, $\tilde{F}e$ maps a natural transformation $\tau : C_{01} \to F$ to the element given by $\tau_X : 1 \to FX$.
2. The above functor $\tilde{F}$ is called the Trnková hull of $F$. It allows us to achieve preservation of intersections for all finitary set functors. Intuitively, a functor on
sets is finitary if its behavior is completely determined by its action on finite sets and functions. For a general functor, this intuition is captured by requiring that the functor preserves filtered colimits [8]. For a set functor $F$ this is equivalent to being finitely bounded, which is the following condition: for each element $x \in FX$ there exists a finite subset $M \subseteq X$ such that $x \in Fi[FM]$, where $i : M \rightarrow X$ is the inclusion map [7, Rem. 3.14].

**Proposition 2.5** [4, p. 66]. The Trnkov hull of a finitary set functor preserves all intersections.

### 2.3 Factorizations.

Recall that an epimorphism $e : A \rightarrow B$ is called strong if it satisfies the following diagonal fill-in property: given a monomorphism $m : C \rightarrow D$ and morphisms $f : A \rightarrow C$ and $g : B \rightarrow D$ such that $m \cdot f = g \cdot e$ then there exists a unique $d : B \rightarrow C$ such that $f = d \cdot e$ and $g = m \cdot d$.

Every complete and well-powered category has factorizations of morphisms: every morphism $f$ may be written as $f = m \cdot e$, where $e$ is a strong epimorphism and $m$ is a monomorphism [9, Prop. 4.4.3]. We call the subobject $m$ the image of $f$. It follows from a result in Kurz’ thesis [16, Prop. 1.3.6] that factorizations of morphisms lift to coalgebras:

**Proposition 2.6 (Coalg $F$ inherits factorizations from $\mathcal{A}$).** Suppose that $F$ preserves monomorphisms. Then the category $\text{Coalg}_F$ has factorizations of homomorphisms $f$ as $f = m \cdot e$, where $e$ is carried by a strong epimorphism and $m$ by a monomorphism in $\mathcal{A}$. The diagonal fill-in property holds in $\text{Coalg}_F$.

**Remark 2.7.** By a subcoalgebra of a coalgebra $(A, \alpha)$ we mean a subobject in $\text{Coalg}_F$ represented by a homomorphism $m : (B, \beta) \rightarrow (A, \alpha)$, where $m$ is monic in $\mathcal{A}$. Similarly, by a strong quotient of a coalgebra $(A, \alpha)$ we mean one represented by a homomorphism $e : (A, \alpha) \rightarrow (C, \gamma)$ with $e$ strongly epic in $\mathcal{A}$.

### 2.4 Chains.

By a transfinite chain in a category $\mathcal{A}$ we understand a functor from the ordered class Ord of all ordinals into $\mathcal{A}$. Moreover, for an ordinal $\lambda$, a $\lambda$-chain in $\mathcal{A}$ is a functor from $\lambda$ to $\mathcal{A}$. A category has colimits of chains if for every ordinal $\lambda$ it has a colimit of every $\lambda$-chain. This includes the initial object $0$ (the case $\lambda = 0$).

**Definition 2.8.** (1) A category $\mathcal{A}$ has smooth monomorphisms if for every $\lambda$-chain $C$ of monomorphisms a colimit exists, its colimit cocone is formed by monomorphisms, and for every cone of $C$ formed by monomorphisms, the factorizing morphism from colim $C$ is monic. In particular, every morphism from 0 is monic.

(2) $\mathcal{A}$ has universally smooth monomorphisms if $\mathcal{A}$ also has pullbacks, and for every morphism $f : X \rightarrow \text{colim} C$, the functor $\mathcal{A} / \text{colim} C \rightarrow \mathcal{A} / X$ forming pullbacks along $f$ preserves the colimit of $C$. This implies that initial object 0 is strict, i.e. every morphism $f : X \rightarrow 0$ is an isomorphism. Indeed, consider the empty chain ($\lambda = 0$).

**Example 2.9.** (1) Set has universally smooth monomorphisms.
(2) \( \text{Vec}_K \) has smooth monomorphisms, but not universally so because the initial object is not strict.

(3) Categories in which colimits of chains and pullbacks are formed “set-like” have universally smooth monomorphisms. These include the categories of posets, graphs, topological spaces, presheaf categories, and many varieties, such as monoids, groups, and unary algebras.

(4) Every locally finitely presentable category \( \mathcal{A} \) with a strict initial object (see Remark 2.12(1)) has smooth monomorphisms. This follows from [8, Prop. 1.62]. Moreover, since pullbacks commute with colimits of chains, it is easy to prove that colimits of chains are universal using the strictness of 0.

(5) The category \( \text{CPO} \) of complete partial orders does not have smooth monomorphisms. Indeed, consider the \( \omega \)-chain of linearly ordered sets \( A_n = \{0, \ldots, n\} + \{\top\} \) (\( \top \) a top element) with inclusion maps \( A_n \to A_{n+1} \). Its colimit is the linearly ordered set \( \mathbb{N} + \{\top, \top'\} \) of natural numbers with two added top elements \( \top' < \top \). For the sub-cpo \( \mathbb{N} + \{\top\} \), the inclusions of \( A_n \) are monic and form a cocone. But the unique factorizing morphism from the colimit is not monic.

**Notation 2.10.** For every object \( A \) we denote by \( \text{Sub}(A) \) the poset of all subobjects of \( A \) (represented by monomorphisms \( s : S \to A \)), where \( s \leq s' \) if there exists \( i \) with \( s = s' \cdot i \). If \( \mathcal{A} \) has pullbacks we have, for every morphism \( f : A \to B \), the inverse image operator, viz. the monotone map \( \overrightarrow{f} : \text{Sub}(B) \to \text{Sub}(A) \) assigning to a subobject \( s : S \to A \) the subobject of \( B \) obtained by forming the inverse image of \( s \) under \( f \), i.e. the pullback of \( s \) along \( f \).

**Lemma 2.11.** If \( \mathcal{A} \) is complete and well-powered, then \( \overrightarrow{f} \) has a left adjoint given by the (direct) image operator \( \overleftarrow{f} : \text{Sub}(A) \to \text{Sub}(B) \). It maps a subobject \( t : T \to B \) to the subobject of \( A \) given by the image of \( f \cdot t \); in symbols we have \( \overleftarrow{f}(t) \leq s \) iff \( t \leq \overrightarrow{f}(s) \).

**Remark 2.12.** If \( \mathcal{A} \) is a complete and well-powered category, then \( \text{Sub}(A) \) is a complete lattice. Now suppose that \( \mathcal{A} \) has smooth monomorphisms.

(1) In this setting, the unique morphism \( \bot_A : 0 \to A \) is a monomorphism and therefore is the bottom element of the poset \( \text{Sub}(A) \).

(2) Furthermore, a join of a chain in \( \text{Sub}(A) \) is obtained by forming a colimit, in the obvious way.

(3) If \( \mathcal{A} \) has universally smooth monomorphisms, then for every morphism \( f : A \to B \), the operator \( \overleftarrow{f} : \text{Sub}(B) \to \text{Sub}(A) \) preserves unions of chains.

**Remark 2.13.** Recall [1] that every endofunctor \( F \) yields the initial-algebra chain, viz. a transfinite chain formed by the objects \( F^0 \) of \( \mathcal{A} \), as follows: \( F^0 = 0 \), the initial object; \( F^{i+1} = F(F^i(0)) \), and for a limit ordinal \( i \) we take the colimit of the chain \( (F^i(0))_{i \leq i} \). The connecting morphisms \( w_{i,j} : F^0 \to F^j(0) \) are defined by a similar transfinite recursion.
3 Recursive Coalgebras

Assumption 3.1. We work with a standard set theory (e.g. Zermelo-Fraenkel), assuming the Axiom of Choice. In particular, we use transfinite induction on several occasions. (We are not concerned with constructive foundations in this paper.)

Throughout this paper we assume that \( \mathcal{A} \) is a complete and well-powered category \( \mathcal{A} \) and that \( F : \mathcal{A} \to \mathcal{A} \) preserves monomorphisms.

For \( \mathcal{A} = \text{Set} \) the condition that \( F \) preserves monomorphisms may be dropped. In fact, preservation of non-empty monomorphism is sufficient in general (for a suitable notion of non-empty monomorphism) [21, Lemma 2.5], and this holds for every set functor.

The following definition of recursive coalgebras was first given by Osius [22]. Taylor [24] speaks of coalgebras obeying the recursion scheme. Capretta et al. [10] extended the concept to parametrically recursive coalgebra by dualizing completely iterative algebras [20].

Definition 3.2. A coalgebra \( \alpha : A \to FA \) is called recursive if for every algebra \( e : FX \to X \) there exists a unique coalgebra-to-algebra morphism \( e^\uparrow : A \to X \), i.e. a unique morphism such that the square on the left below commutes:

\[
\begin{array}{ccc}
A & \xrightarrow{e^\uparrow} & X \\
\downarrow{\alpha} & & \uparrow{e} \\
FA & \xrightarrow{Fe^\uparrow} & FX
\end{array}
\quad \quad \quad \quad \quad \quad
\begin{array}{ccc}
A & \xrightarrow{e^\uparrow} & X \\
\downarrow{(\alpha,A)} & & \uparrow{e} \\
FA \times A & \xrightarrow{Fe^\uparrow \times A} & FX \times A
\end{array}
\]

\((A, \alpha)\) is called parametrically recursive if for every morphism \( e : FX \times A \to X \) there is a unique morphism \( e^\uparrow : A \to X \) such that the square on the right above commutes.

Example 3.3. (1) A graph regarded as a coalgebra for \( \mathcal{P} \) is recursive iff it has no infinite path. This is an immediate consequence of the General Recursion Theorem (see Corollary 5.6 and Example 4.5(2)).

(2) Let \( \iota : F(\mu F) \to \mu F \) be an initial algebra. By Lambek’s Lemma, \( \iota \) is an isomorphism. So we have a coalgebra \( \iota^{-1} : \mu F \to F(\mu F) \). This algebra is (parametrically) recursive. By [20, Thm. 2.8], in dual form, this is precisely the same as the terminal parametrically recursive coalgebra (see also [10, Prop. 7]).

(3) The initial coalgebra \( 0 \to F0 \) is recursive.

(4) If \( (C, \gamma) \) is recursive so is \( (FC, F\gamma) \), see [10, Prop. 6].

(5) Colimits of recursive coalgebras in \( \text{Coalg} F \) are recursive. This is easy to prove, using that colimits of coalgebras are formed on the level of the underlying category.

(6) It follows from items (3)-(5) that in the initial-algebra chain from Remark 2.13 all coalgebras \( w_{0,i+1} : F^i0 \to F^{i+1}0, i \in \text{Ord} \), are recursive.
(7) Every parametrically recursive coalgebra is recursive. (To see this, form for a given \(\varepsilon: FX \rightarrow X\) the morphism \(\varepsilon = e \cdot \pi\), where \(\pi: FX \times A \rightarrow FX\) is the projection.) In Corollaries 5.6 and 5.9 we will see that the converse often holds.

Here is an example where the converse fails [3]. Let \(R: \text{Set} \rightarrow \text{Set}\) be the functor defined in Example 2.2(4). Also, let \(C = \{0, 1\}\), and define \(\gamma: C \rightarrow RC\) by \(\gamma(0) = \gamma(1) = (0, 1)\). Then \((C, \gamma)\) is a recursive coalgebra. Indeed, for every algebra \(\alpha: RA \rightarrow A\) the constant map \(h: C \rightarrow A\) with \(h(0) = h(1) = \alpha(d)\) is the unique coalgebra-to-algebra morphism.

However, \((C, \gamma)\) is not parametrically recursive. To see this, consider any morphism \(e: RX \times \{0, 1\} \rightarrow X\) such that \(RX\) contains more than one pair \((x_0, x_1), x_0 \neq x_1\) with \(e((x_0, x_1), i) = x_i\) for \(i = 0, 1\). Then each such pair yields \(h: C \rightarrow X\) with \(h(i) = x_i\) making the appropriate square commutative. Thus, \((C, \gamma)\) is not parametrically recursive.

(8) Capretta et al. [11] showed that recursivity semantically models divide-and-conquer programs, as demonstrated by the example of Quicksort. For every linearly ordered set \(A\) (of data elements), Quicksort is usually defined as the recursive function \(q: A^* \rightarrow A^*\) given by

\[ q(\varepsilon) = \varepsilon \quad \text{and} \quad q(aw) = q(w_{\leq a}) \star (aq(w_{>a})), \]

where \(A^*\) is the set of all lists on \(A\), \(\varepsilon\) is the empty list, \(\star\) is the concatenation of lists and \(w_{\leq a}\) denotes the list of those elements of \(w\) which are less than or equal than \(a\); analogously for \(w_{>a}\).

Now consider the functor \(FX = 1 + A \times X \times X\) on \(\text{Set}\), where \(1 = \{\bullet\}\), and form the coalgebra \(s: A^* \rightarrow 1 + A \times A^* \times A^*\) given by

\[ s(\varepsilon) = \bullet \quad \text{and} \quad s(aw) = (a, w_{\leq a}, w_{>a}) \quad \text{for} \ a \in A \ \text{and} \ w \in A^*. \]

We shall see that this coalgebra is recursive in Example 5.3. Thus, for the \(F\)-algebra \(m: 1 + A \times A^* \times A^* \rightarrow A^*\) given by

\[ m(\bullet) = \varepsilon \quad \text{and} \quad m(a, w, v) = w \star (av) \]

there exists a unique function \(q\) on \(A^*\) such that \(q = m \cdot Fq \cdot s\). Notice that the last equation reflects the idea that Quicksort is a divide-and-conquer algorithm. The coalgebra structure \(s\) divides a list into two parts \(w_{\leq a}\) and \(w_{>a}\). Then \(Fq\) sorts these two smaller lists, and finally in the combine- (or conquer-) step, the algebra structure \(m\) merges the two sorted parts to obtain the desired whole sorted list.

Jeannin et al. [15, Sec. 4] provide a number of recursive functions arising in programming that are determined by recursivity of a coalgebra, e.g. the gcd of integers, the Ackermann function, and the Towers of Hanoi.

4 The Next Time Operator and Well-Founded Coalgebras

As we have mentioned in the Introduction, the main issue of this paper is the relationship between two concepts pertaining to coalgebras: recursiveness and
well-foundedness. The concept of well-foundedness is well-known for directed
graphs \((G, \rightarrow)\): it means that there are no infinite directed paths \(g_0 \rightarrow g_1 \rightarrow \cdots\).
For a set \(X\) with a relation \(R\), well-foundedness means that there are no \(\text{backwards}\)
sequences \(\cdots R x_2 R x_1 R x_0\), i.e. the converse of the relation is well-founded as a
graph. Taylor [24, Def. 6.2.3] gave a more general category theoretic formulation
of well-foundedness. We observe here that his definition can be presented in a
compact way, by using an operator that generalizes the way one thinks of the
semantics of the ‘next time’ operator of temporal logics for non-deterministic (or
even probabilistic) automata and transitions systems. It is also strongly related
to the algebraic semantics of modal logic, where one passes from a graph \(G\)
to a function on \(\mathcal{P}G\). Jacobs [14] defined and studied the ‘next time’ operator
on coalgebras for Kripke polynomial set functors. This can be generalized to
arbitrary functors as follows.

Recall that \(\text{Sub}(A)\) denotes the complete lattice of subobjects of \(A\).

**Definition 4.1** [4, Def. 8.9]. Every coalgebra \(\alpha : A \rightarrow FA\) induces an endo-
function on \(\text{Sub}(A)\), called the \textit{next time operator}

\[
\bigcirc : \text{Sub}(A) \rightarrow \text{Sub}(A), \quad \bigcirc(s) = \overline{\alpha}(Fs) \quad \text{for } s \in \text{Sub}(A).
\]

In more detail: we define \(\bigcirc s\) and \(\alpha(s)\) by the pullback in (4.1). (Being a pullback is
indicated by the “corner” symbol.) In words, \(\bigcirc\) assigns to each subobject \(s : S \rightarrow A\) the inverse image
of \(Fs\) under \(\alpha\). Since \(Fs\) is a monomorphism, \(\bigcirc s\) is a
monomorphism and \(\alpha(s)\) is (for every representation
\(\bigcirc s\) of that subobject of \(A\) uniquely determined.

**Example 4.2.** (1) Let \(A\) be a graph, considered as a coalgebra for \(\mathcal{P} : \text{Set} \rightarrow \text{Set}\).
If \(S \subseteq A\) is a set of vertices, then \(\bigcirc S\) is the set of vertices all of whose successors
belong to \(S\).

(2) For the set functor \(FX = \mathcal{P}(\Sigma \times X)\) expressing labelled transition systems
the operator \(\bigcirc\) for a coalgebra \(\alpha : A \rightarrow \mathcal{P}(\Sigma \times A)\) is the semantic counterpart
of the next time operator of classical linear temporal logic, see e.g. Manna and
Pnueli [18]. In fact, for a subset \(S \subseteq A\) we have that \(\bigcirc S\) consists of those states
all of whose next states lie in \(S\), in symbols:

\[
\bigcirc S = \{ x \in A \mid (s, y) \in \alpha(x) \text{ implies } y \in S, \text{ for all } s \in \Sigma\}.
\]

The next time operator allows a compact definition of well-foundedness as
characterized by Taylor [24, Exercise VI.17] (see also [6, Corollary 2.19]):

**Definition 4.3.** A coalgebra is \textit{well-founded} if \(id_A\) is the only fixed point of its
next time operator.

**Remark 4.4.** (1) Let us call a subcoalgebra \(m : (B, \beta) \rightarrow (A, \alpha)\) \textit{cartesian}
provided that the square (4.2) is a pullback. Then
\((A, \alpha)\) is well-founded iff it has no proper cartesian
subcoalgebra. That is, if \(m : (B, \beta) \rightarrow (A, \alpha)\) is a
cartesian subcoalgebra, then \(m\) is an isomorphism.
Indeed, the fixed points of next time are precisely the

\[
\begin{array}{ccc}
B & \xrightarrow{\beta} & FB \\
\downarrow m & & \downarrow \varphi_m \\
A & \xrightarrow{\alpha} & FA
\end{array}
\]
cartesian subcoalgebras.

(2) A coalgebra is well-founded iff $\bigcirc$ has a unique pre-fixed point $\bigcirc m \leq m$. Indeed, since $\text{Sub}(A)$ is a complete lattice, the least fixed point of a monotone map is its least pre-fixed point. Taylor's definition [24, Def. 6.3.2] uses that property: he calls a coalgebra well-founded iff $\bigcirc$ has no proper subobject as a pre-fixed point.

**Example 4.5.** (1) Consider a graph as a coalgebra $\alpha: A \to \mathcal{P}A$ for the powerset functor (see Example 2.1). A subcoalgebra is a subset $m: B \hookrightarrow A$ such that with every vertex $v$ it contains all neighbors of $v$. The coalgebra structure $\beta: B \to \mathcal{P}B$ is then the domain-codomain restriction of $\alpha$. To say that $B$ is a cartesian subcoalgebra means that whenever a vertex of $A$ has all neighbors in $B$, it also lies in $B$. It follows that $(A, \alpha)$ is well-founded iff it has no infinite directed path, see [24, Example 6.3.3].

(2) If $\mu F$ exists, then as a coalgebra it is well-founded. Indeed, in every pullback (4.2), since $\epsilon^{-1}$ (as $\alpha$) is invertible, so is $\beta$. The unique algebra homomorphism from $\mu F$ to the algebra $\beta^{-1}: FB \to B$ is clearly inverse to $m$.

(3) If a set functor $F$ fulfills $F\emptyset = \emptyset$, then the only well-founded coalgebra is the empty one. Indeed, this follows from the fact that the empty coalgebra is a fixed point of $\bigcirc$. For example, a deterministic automaton over the input alphabet $\Sigma$, as a coalgebra for $FX = \{0,1\} \times X^{\Sigma}$, is well-founded iff it is empty.

(4) A non-deterministic automaton may be considered as a coalgebra for the set functor $FX = \{0,1\} \times (\mathcal{P}X)^{\Sigma}$. It is well-founded iff the state transition graph is well-founded (i.e., has no infinite path). This follows from Corollary 4.10 below.

(5) A linear weighted automaton, i.e. a coalgebra for $FX = K \times X^{\Sigma}$ on $\text{Vec}_K$, is well-founded iff every path in its state transition graph eventually leads to 0. This means that every path starting in a given state leads to the state 0 after finitely many steps (where it stays).

**Notation 4.6.** Given a set functor $F$, we define for every set $X$ the map $\tau_X: FX \to \mathcal{P}X$ assigning to every element $x \in FX$ the intersection of all subsets $m: M \hookrightarrow X$ such that $x$ lies in the image of $Fm$:

$$\tau_X(x) = \bigcap \{m \mid m: M \hookrightarrow X \text{ satisfies } x \in Fm[FM]\}. \quad (4.3)$$

Recall that a functor preserves intersections if it preserves (wide) pullbacks of families of monomorphisms.

Gumm [13, Thm. 7.3] observed that for a set functor preserving intersections, the maps $\tau_X: FX \to \mathcal{P}X$ in (4.3) form a “subnatural” transformation from $F$ to the powerset functor $\mathcal{P}$. Subnaturality means that (although these maps do not form a natural transformation in general) for every monomorphism $i: X \to Y$ we have a commutative square:

$$
\begin{array}{ccc}
FX & \xrightarrow{\tau_X} & \mathcal{P}X \\
F\downarrow & & \downarrow \mathcal{P}i \\
FY & \xrightarrow{\tau_Y} & \mathcal{P}Y
\end{array}
$$

(4.4)
Remark 4.7. As shown in [13, Thm. 7.4] and [23, Prop. 7.5], a set functor $F$ preserves intersections if the squares in (4.4) above are pullbacks. Moreover, loc. cit. and [13, Thm. 8.1] prove that $\tau: F \to \mathcal{P}$ is a natural transformation, provided $F$ preserves inverse images and intersections.

Definition 4.8. Let $F$ be a set functor. For every coalgebra $\alpha: A \to FA$ its canonical graph is the following coalgebra for $\mathcal{P}: A \xrightarrow{\alpha} FA \xrightarrow{\tau_A} \mathcal{P}A$.

Thanks to the subnaturality of $\tau$ one obtains the following results.

Proposition 4.9. For every set functor $F$ preserving intersections, the next time operator of a coalgebra $(A, \alpha)$ coincides with that of its canonical graph.

Corollary 4.10 [24, Rem. 6.3.4]. A coalgebra for a set functor preserving intersections is well-founded iff its canonical graph is well-founded.

Example 4.11. (1) For a (deterministic or non-deterministic) automaton, the canonical graph has an edge from $s$ to $t$ iff there is a transition from $s$ to $t$ for some input letter. Thus, we obtain the characterization of well-foundedness as stated in Example 4.5(3) and (4).

(2) Every polynomial functor $H_N: \text{Set} \to \text{Set}$ preserves intersections. Thus, a coalgebra $(A, \alpha)$ is well-founded if there are no infinite paths in its canonical graph. The canonical graph of $A$ has an edge from $a$ to $b$ if $\alpha(a)$ is of the form $\sigma(c_1, \ldots, c_n)$ for some $\sigma \in \Sigma_n$ and if $b$ is one of the $c_i$’s.

(3) Thus, for the functor $FX = 1 + A \times X \times X$, the coalgebra $(A^*, s)$ of Example 3.3(8) is easily seen to be well-founded via its canonical graph. Indeed, this graph has for every list $w$ one outgoing edge to the list $w_{\leq a}$ and one to $w_{>a}$ for every $a \in A$. Hence, this is a well-founded graph.

Lemma 4.12. The next time operator is monotone: if $m \leq n$, then $\ominus m \leq \ominus n$.

Lemma 4.13. Let $\alpha: A \to FA$ be a coalgebra and $m: B \to A$ a subobject.

(1) There is a coalgebra structure $\beta: B \to FB$ for which $m$ gives a subcoalgebra of $(A, \alpha)$ iff $m \leq \ominus m$.

(2) There is a coalgebra structure $\beta: B \to FB$ for which $m$ gives a cartesian subcoalgebra of $(A, \alpha)$ iff $m = \ominus m$.

Lemma 4.14. For every coalgebra homomorphism $f: (B, \beta) \to (A, \alpha)$ we have

$$\ominus \beta \cdot f \leq f \cdot \ominus \alpha,$$

where $\ominus \alpha$ and $\ominus \beta$ denote the next time operators of the coalgebras $(A, \alpha)$ and $(B, \beta)$, respectively, and $\leq$ is the pointwise order.

Corollary 4.15. For every coalgebra homomorphism $f: (B, \beta) \to (A, \alpha)$ we have $\ominus \beta \cdot f = f \cdot \ominus \alpha$, provided that either

- $f(\beta(a)) \subseteq \ominus \alpha(a)$ for each $a \in A$,
- $\beta(\ominus \alpha(a)) \subseteq f(\beta(a))$ for each $a \in A$.
(1) $f$ is a monomorphism in $\mathcal{A}$ and $F$ preserves finite intersections, or
(2) $F$ preserves inverse images.

**Definition 4.16 [4].** The well-founded part of a coalgebra is its largest well-founded subcoalgebra.

The well-founded part of a coalgebra always exists and is the coreflection in the category of well-founded coalgebras [6, Prop. 2.27]. We provide a new, shorter proof of this fact. The well-founded part is obtained by the following:

**Construction 4.17 [6, Not. 2.22].** Let $\alpha: A \to FA$ be a coalgebra. We know that $\text{Sub}(A)$ is a complete lattice and that the next time operator $\Box$ is monotone (see Lemma 4.12). Hence, by the Knaster-Tarski fixed point theorem, $\Box$ has a least fixed point, which we denote by $a^*: A^* \to A$.

By Lemma 4.13(2), we know that there is a coalgebra structure $\alpha^*: A^* \to FA^*$ so that $a^*: (A^*, \alpha^*) \to (A, \alpha)$ is the smallest cartesian subcoalgebra of $(\overline{\Box}, \alpha)$.

**Proposition 4.18.** For every coalgebra $(A, \alpha)$, the coalgebra $(A^*, \alpha^*)$ is well-founded.

**Proof.** Let $m: (B, \beta) \to (A^*, \alpha^*)$ be a cartesian subcoalgebra. By Lemma 4.13, $a^* \cdot m: B \to A$ is a fixed point of $\Box$. Since $a^*$ is the least fixed point, we have $a^* \leq a^* \cdot m$, i.e. $a^* = a^* \cdot m \cdot x$ for some $x: A^* \to B$. Since $a^*$ is monic, we thus have $m \cdot x = id_{A^*}$. So $m$ is a monomorphism and a split epimorphism, whence an isomorphism. $\Box$

**Proposition 4.19.** The full subcategory of $\text{Coalg}$ $F$ given by well-founded coalgebras is coreflective. In fact, the well-founded coreflection of a coalgebra $(A, \alpha)$ is its well-founded part $a^*: (A^*, \alpha^*) \to (A, \alpha)$.

**Proof.** We are to prove that for every coalgebra homomorphism $f: (B, \beta) \to (A, \alpha)$, where $(B, \beta)$ is well-founded, there exists a coalgebra homomorphism $f^*: (B, \beta) \to (A^*, \alpha^*)$ such that $a^* \cdot f^* = f$. The uniqueness is easy.

For the existence of $f^*$, we first observe that $\overline{\Box}(a^*)$ is a pre-fixed point of $\Box_B$; indeed, using Lemma 4.14 we have $\Box_B(\overline{\Box}(a^*)) \leq \overline{\Box}(\Box_B(a^*)) = \overline{\Box}(a^*)$.

By Remark 4.4(2), we therefore have $id_B = b^* \leq \overline{\Box}(a^*)$ in Sub($B$). Using the adjunction of Lemma 2.11, we have $\overline{\Box}(id_B) \leq a^*$ in Sub($A$). Now factorize $f$ as $B \twoheadrightarrow C \rightarrowtail A$. We have $\overline{\Box}(id_B) = m$, and we then obtain $m = \overline{\Box}(id_B) \leq a^*$, i.e. there exists a morphism $h: C \to A^*$ such that $a^* \cdot h = m$. Thus, $f^* = h \cdot e: B \to A^*$ is a morphism satisfying $a^* \cdot f^* = a^* \cdot h \cdot e = m \cdot e = f$. It follows that $f^*$ is a coalgebra homomorphism from $(B, \beta)$ to $(A^*, \alpha^*)$ since $f$ and $a^*$ are and $F$ preserves monomorphisms. $\Box$

**Construction 4.20 [6, Not. 2.22].** Let $(A, \alpha)$ be a coalgebra. We obtain $a^*$, the least fixed point of $\Box$, as the join of the following transfinite chain of subobjects $a_i: A_i \to A$, $i \in \text{Ord}$. First, put $a_0 = \bot_A$, the least subobject of $A$. Given $a_i: A_i \to A$, put $a_{i+1} = \bigcup a_i: A_{i+1} = \bigcup A_i \to A$. For every limit ordinal $\lambda$, put $a_\lambda = \bigcup_{i < \lambda} a_i$. Since Sub($A$) is a set, there exists an ordinal $i$ such that $a_i = a^*: A^* \to A$. 

Remark 4.21. Note that, whenever monomorphisms are smooth, we have $A_0 = 0$ and the above join $a_j$ is obtained as the colimit of the chain of the subobject $a_i: A_i \to A$, $i < j$ (see Remark 2.12).

If $F$ is a finitary functor on a locally finitely presentable category, then the least ordinal $i$ with $a^* = a_i$ is at most $\omega$, but in general one needs transfinite iteration to reach a fixed point.

Example 4.22. Let $(A, \alpha)$ be a graph regarded as a coalgebra for $\mathcal{P}$ (see Example 2.1). Then $A_0 = \emptyset$, $A_1$ is formed by all leaves; i.e. those nodes with no neighbors, $A_2$ by all leaves and all nodes such that every neighbor is a leaf, etc. We see that a node $x$ lies in $A_{i+1}$ iff every path starting in $x$ has length at most $i$. Hence $A^* = A_\omega$ is the set of all nodes from which no infinite paths start.

We close with a general fact on well-founded parts of fixed points (i.e. (co)algebras whose structure is invertible). The following result generalizes [15, Cor. 3.4], and it also appeared before for functors preserving finite intersections [4, Theorem 8.16 and Remark 8.18]. Here we lift the latter assumption (see [5, Theorem 7.6] for the new proof):

Theorem 4.23. Let $\mathcal{A}$ be a complete and well-powered category with smooth monomorphisms. For $F$ preserving monomorphisms, the well-founded part of every fixed point is an initial algebra. In particular, the only well-founded fixed point is the initial algebra.

Example 4.24. We illustrate that for a set functor $F$ preserving monomorphisms, the well-founded part of the terminal coalgebra is the initial algebra. Consider $FX = A \times X + 1$. The terminal coalgebra is the set $A^\infty \cup A^*$ of finite and infinite sequences from the set $A$. The initial algebra is $A^*$. It is easy to check that $A^*$ is the well-founded part of $A^\infty \cup A^*$.

5 The General Recursion Theorem and its Converse

The main consequence of well-foundedness is parametric recursivity. This is Taylor’s General Recursion Theorem [24, Theorem 6.3.13]. Taylor assumed that $F$ preserves inverse images. We present a new proof for which it is sufficient that $F$ preserves monomorphisms, assuming those are smooth.

Theorem 5.1 (General Recursion Theorem). Let $\mathcal{A}$ be a complete and wellpowered category with smooth monomorphisms. For $F: \mathcal{A} \to \mathcal{A}$ preserving monomorphisms, every well-founded coalgebra is parametrically recursive.

Proof sketch. (1) Let $(A, \alpha)$ be well-founded. We first prove that it is recursive. We use the subobjects $a_i: A_i \to A$ of Construction 4.20\footnote{One might object to this use of transfinite recursion, since Theorem 5.1 itself could be used as a justification for transfinite recursion. Let us emphasize that we are not presenting Theorem 5.1 as a foundational contribution. We are building on the classical theory of transfinite recursion.}, the corresponding
morphisms $\alpha(a_i): A_{i+1} = \bigcirc A_i \to FA_i$ (cf. Definition 4.3), and the recursive coalgebras $(F^0, w_{i,i+1})$ of Example 3.3(6). We obtain a natural transformation $h$ from the chain $(A_i)$ in Construction 4.20 to the initial-algebra chain $(F^0)$ (see Remark 2.13) by transfinite recursion.

Now for every algebra $e: FX \to X$, we obtain a unique coalgebra-to-algebra morphism $f_i: F^i 0 \to X$, i.e. we have that $f_i = e \cdot F f_i \cdot w_{i,i+1}$. Since $(A, \alpha)$ is well-founded, we know that $\alpha = \alpha^* = \alpha(a_i)$ for some $i$. From this it is not difficult to prove that $f_i \cdot h_i$ is a coalgebra-to-algebra morphism from $(A, \alpha)$ to $(X, e)$.

In order to prove uniqueness, we prove by transfinite induction that for any given coalgebra-to-algebra homomorphism $e^\downarrow$, one has $e^\downarrow \cdot a_j = f_j \cdot h_j \cdot a_j$ for every ordinal number $j$. Then for the above ordinal number $i$ with $a_i = id_A$, we have $e^\downarrow = f_i \cdot h_i$, as desired. This shows that $(A, \alpha)$ is recursive.

(2) We prove that $(A, \alpha)$ is parametrically recursive. Consider the coalgebra $(\alpha, id_A): A \to FA \times A$ for $F(-) \times A$. This functor preserves monomorphisms since $F$ does and monomorphisms are closed under products. The next time operator $\bigcirc$ on $\text{Sub}(A)$ is the same for both coalgebras since the square (4.1) is a pullback if and only if the square on the right below is one.

Since $id_A$ is the unique fixed point of $\bigcirc$ w.r.t. $F$ (see Definition 4.3), it is also the unique fixed point of $\bigcirc$ w.r.t. $F(-) \times A$. Thus, $(A, (\alpha, id_A))$ is a well-founded coalgebra for $F(-) \times A$. By the previous argument, this coalgebra is thus recursive for $F(-) \times A$; equivalently, $(A, \alpha)$ is parametrically recursive for $F$. \hfill \Box

**Theorem 5.2.** For every endofunctor on $\text{Set}$ or $\text{Vec}_K$ (vector spaces and linear maps), every well-founded coalgebra is parametrically recursive.

**Proof sketch.** For $\text{Set}$, we apply Theorem 5.1 to the Trnková hull $\tilde{F}$ (see Proposition 2.3), noting that $F$ and $\tilde{F}$ have the same (non-empty) coalgebras. Moreover, one can show that every well-founded (or recursive) $F$-coalgebra is a well-founded (recursive, resp.) $\tilde{F}$-coalgebra. For $\text{Vec}_K$, observe that monomorphisms split and are therefore preserved by every endofunctor $F$. \hfill \Box

**Example 5.3.** We saw in Example 4.11(3) that for $FX = 1 + A \times X \times X$ the coalgebra $(A, s)$ from Example 3.3(8) is well-founded, and therefore it is (parametrically) recursive.

**Example 5.4.** Well-founded coalgebras need not be recursive when $F$ does not preserve monomorphisms. We take $\mathcal{A}$ to be the category of sets with a predicate, i.e. pairs $(X, A)$, where $A \subseteq X$. Morphisms $f: (X, A) \to (Y, B)$ satisfy $f[A] \subseteq B$. Denote by $1$ the terminal object $(1, 1)$. We define an endofunctor $F$ by $F(X, \emptyset) = (X + 1, \emptyset)$, and for $A \neq \emptyset$, $F(X, A) = 1$. For a morphism $f: (X, A) \to (Y, B)$, put $Ff = f + id$ if $A = \emptyset$; if $A \neq \emptyset$, then also $B \neq \emptyset$ and $Ff$ is $id: 1 \to 1$. 