EPTCS 430

Proceedings of the

Twentieth International Symposium on

Logical and Semantic Frameworks with Applications

Brasilia, Brazil, October 7-8, 2025

Edited by: Haniel Barbosa and Christophe Ringeissen

Published: 2nd October 2025
DOI: 10.4204/EPTCS.430
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Preface

This volume contains the proceedings of the 20th Workshop on Logical and Semantic Frameworks with Applications (LSFA 2025), which was held in Brasília, the capital of Brazil, from October 7 to October 8, 2025.

The aim of the LSFA series of workshops is bringing together theoreticians and practitioners to promote new techniques and results, from the theoretical side, and feedback on the implementation and use of such techniques and results, from the practical side. LSFA includes areas such as proof and type theory, equational deduction and rewriting systems, automated reasoning and concurrency theory.

The 20th International Workshop on Logical and Semantic Frameworks with Applications continues the successful workshops held in Natal (2006), Ouro Preto (2007), Salvador (2008), Brasília (2009), Natal (2010), Belo Horizonte (2011), Rio de Janeiro and Niterói (2012), São Paulo (2013), Brasília (2014), Natal (2015), Porto (2016), Brasília (2017), Fortaleza (2018), Natal (2019), Bahia (2020), Buenos Aires (virtual) (2021), Belo Horizonte (2022), Rome (2023), and Goiânia (2024).

For LSFA 2025, seven contributions were accepted out of eleven submissions, with three or four reviewers per submission. Among the seven accepted submissions, five regular papers were accepted for inclusion in this volume and two submissions were accepted as presentation-only contributions.

In addition, the program included three invited talks given by Temur Kutsia (Research Institute for Symbolic Computation, Johannes Kepler University), Bruno Lopes (Instituto de Computação, Universidade Federal Fluminense), and Yoni Zohar (Department of Computer Science, Bar Ilan University).

Temur Kutsia's presentation was a joint invited talk with CICM 2025.

Many people helped to make LSFA 2025 a successful event. We would like to thank all the PC members, the additional subreviewers, and the LSFA 2025 organizers.

Haniel Barbosa, Christophe Ringeissen (LSFA 2025 PC co-chairs)

Program Committee

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. ii–iii, doi:10.4204/EPTCS.430.0

- Marcelo Finger, University of São Paulo, Brazil
- Mário Florido, FCUP, Portugal
- Pascal Fontaine, Université de Liège, Belgium
- Delia Kesner, Université Paris Cité, France
- Alberto Momigliano, Università degli Studi di Milano, Italy
- Flavio L. C. de Moura, Universidade de Brasília, Brazil
- Cláudia Nalon, Universidade de Brasília, Brazil
- Carlos Olarte, LIPN/Université Sorbonne Paris Nord, France
- Elaine Pimentel, University College London, UK
- Christophe Ringeissen, Inria - LORIA, France (PC co-chair)
- Camilo Rocha, Pontificia Universidad Javeriana Cali, Colombia
- Helida Santos, FURG, Brazil
- Mallku Soldevila, Universidade Federal de Minas Gerais, Brazil
- Deivid Vale, Augusta University, USA
- Daniel Ventura, Universidade Federal de Goiás, Brazil

Additional Subreviewers

- Murdoch Gabbay
- Dominique Larchey-Wendling
- Pedro Saccomani
- Niels van der Weide

Organizing Committee

- Andréia B. Avelar, Universidade de Brasília, Brazil
- Mauricio Ayala-Rincón, Universidade de Brasília, Brazil
- André Luiz Galdino, Universidade Federal de Catalão, Brazil
- Flavio L. C. de Moura, Universidade de Brasília, Brazil
- Cláudia Nalon, Universidade de Brasília, Brazil
- Daniele Nantes-Sobrinho, Imperial College, UK, and Universidade de Brasília, Brazil

Solving Symbolic Constraints

Temur Kutsia
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University

Symbolic constraint solving is a fundamental technique in many areas of mathematics and computer science, providing the basis for key operations in automated reasoning, term rewriting, declarative programming, or formal methods. Important classes of symbolic constraints include unification, matching, generalization, disunification, ordering constraints. Efficient methods for solving them form the computational core of inference engines, proof assistants, program transformation systems, logical frameworks, etc.

In this talk, we present recent advances in solving unification, matching, and generalization constraints in expressive theories, including those that involve background knowledge or binding constructs. We discuss algorithmic techniques for constraint solving, identify classes of problems that admit efficient solutions, and highlight their relevance for tools and techniques in symbolic computation and computer mathematics.

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. iv–iv, doi:10.4204/EPTCS.430.0.1

A Note on Propositional Dynamic Logic Expressiveness and Complexity

Bruno Lopes
Instituto de Computação
Universidade Federal Fluminense

Propositional Dynamic Logic (PDL) is a sophisticated modal logic tailored to reason about programs. Several fragments and extensions of PDL are worth studying, allowing us to control complexity or increase expressiveness.

By limiting PDL to deterministic fragments, such as Deterministic PDL or Strictly Deterministic PDL, we introduce new program behaviors by excluding nondeterministic choices or restricting the form of iteration. These restrictions affect both expressiveness and computational complexity. While standard PDL is decidable and has an EXPTIME-complete satisfiability, some deterministic fragments have lower complexity bounds, such as PSPACE-complete. It is a showcase of a trade-off between expressiveness and tractability.

Conversely, attempts to increase expressiveness come with significant challenges. Extensions like Context-Free PDL result in undecidable validity. This discussion will explore the effects of restricting to deterministic programs and examine how to enhance expressiveness using memory.

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. v–v, doi:10.4204/EPTCS.430.0.2

My Attempts To Save Politeness

Yoni Zohar
Department of Computer Science
Bar Ilan University

Theory combination in Satisfiability Modulo Theories (SMT) deals with the generation of algorithms for combinations of theories, based on the algorithms for each of the specific theories being combined.

Among the most important combination methods is the polite combination method, which has two major advantages: first, it only requires solvers for the theories being combined, and no other mechanism is needed (e.g., there is no need for an algorithm that computes cardinalities of models). Second, it only requires one of the theories to admit some property, while the other only has to be decidable.

When polite combination was introduced, it was claimed that the required property is *politeness* (a notion I will formally and intuitively describe in my talk). But, later, it was shown that the correctness proof had a bug, and for the proof to work, a stronger property is needed to be assumed, namely *strong politeness*.

Hence, while the importance of the polite combination *method* is unquestionable (its incorporation in the state-of-the-art solver cvc5 is a case in point), it was left unclear what is the value of the politeness *property*. Can it still be considered a property that is relevant to theory combination?

In this talk I will describe several attempts made by myself and others to answer this question positively, thus saving the politeness property.

No background in theory combination or SMT is assumed, though politeness is expected.

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. vi–vi, doi:10.4204/EPTCS.430.0.3

Towards the HSP Theorem in Nominal Algebra with Fixed-Point Constraints (Work in Progress)

Ali K. Caires-Santos
Departament of Mathematics, University of Brasília, Brasília, Brazil
A.K.C.R.Santos@mat.unb.br

Reasoning about equality modulo equational theories in languages with variable binding requires careful handling of names, freshness, and α-equivalence. The nominal approach provides a solid framework with applications in unification, rewriting, and universal algebra. Recent work replaces traditional freshness constraints with permutation fixed-point constraints scoped by the quantifier M, ranging over fresh names. Based on this framework, we define a proof system and discuss its soundness and completeness with respect to nominal set semantics. Unlike in the original nominal algebra, this system does not satisfy the classical HSP theorem because homomorphic images may fail to preserve fixed-points. We explore restricting homomorphisms to those preserving fixed-points to recover an HSP-like result. This talk presents these ideas and discusses open problems.

Overview

Equality plays a central role in algebra and logic, but in the λ-calculus it is complicated not only by variable binding and freshness conditions but also by the higher-order nature of the system and in extensions such as the η-rule. For example, the η-rule ($\lambda x.(Mx) = M$ when x is not free in M) relies on distinguishing bound and free variables, as well as between object-level (like x) and meta-level variables (like M).

To address such challenges, three major paradigms for representing bindings have been developed: De Bruijn indices, higher-order abstract syntax (HOAS), and the *nominal* approach. Among these, the nominal approach introduced by Gabbay and Pitts [6] provides an explicit and flexible treatment of names and binding, supporting α-equivalence, freshness, and name generation in a way that aligns closely with mathematical practice. Since then, nominal techniques have been applied across several domains such as unification [7], rewriting [3], and universal algebra [5].

In these domains, nominal terms play a central role by presenting a two-level syntax constructed from a countably infinite set of *atoms* $\mathbb{A} = \{a, b, c, \ldots\}$ representing object-level variables, a countably infinite set of unknowns $\mathbb{V} = \{X, Y, Z, \ldots\}$ representing meta-level variables, function symbols $\mathtt{f}$, and name abstractions $[a]t$. The abstraction $[a]t$ models the binding of the atom a in the term t, similar to variable binding in expressions like $\lambda x.t$ or $\forall x.\phi$. Formally, terms are given by the grammar:

$$t, u ::= a \mid \pi \cdot X \mid \mathtt{f}(t_1, \ldots, t_n) \mid [a]t$$

where $\pi \cdot X$ is a *suspension* applying a finite permutation π from the group $\mathscr{G}(\mathbb{A})$ of atom permutations to a meta-variable X. When $\pi = \mathtt{id}$, we simply write X. Intuitively, suspensions can be read as "permute as π in whatever X becomes". The framework also includes *freshness* constraints, denoted $a\#t$, which generalize the notion of "does not occur free". When $t \equiv X$, such constraints are called *primitives*.

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. vii–ix, doi:10.4204/EPTCS.430.0.4

Nominal universal algebra [5] builds on the syntax of nominal terms to define a proof system for reasoning about equality modulo an equational theory $\mathtt{T}$, while incorporating α-equivalence. It also studies the models of this reasoning through *nominal sets*: sets equipped with an action of the group $\mathscr{G}(\mathbb{A})$, where each element x has a finite set of atoms it depends on, called its *support*, denoted by $\mathtt{supp}(x)$. Freshness constraints are interpreted semantically by the relation $a\#_{\mathtt{sem}}x$, which holds if and only if $a \notin \mathtt{supp}(x)$. Alternatively, freshness can be intuitively understood as a combination of the quantifier $И$ and fixed-point equalities under atom swapping. Formally, semantic freshness can be expressed as (see [6, Eq. 13]): $a\#_{\mathtt{sem}}x$ if and only if $И\mathbf{c}.(a\ \mathbf{c})\cdot x = x$, where the quantifier $И\mathbf{c}.P(\mathbf{c})$ means that the property $P(\mathbf{c})$ holds for *all but finitely many* atoms $\mathbf{c}$.

Semantic freshness can be seen as a negative condition: $a\#_{\mathtt{sem}}x$ means that the atom a is not in the support of x. Fixed points rephrase this in a more positive way: an atom a is fresh for x when it stays the same after swapping a with all but finitely many other atoms. This shift is interesting because many properties we care about in algebra and logic, such as commutativity, are described as invariances, not absences.

Seeing freshness as invariance explains *why* a is fresh for an element, instead of just stating it. For example, a is fresh for $x = \mathtt{f}(b,d)$ not only because a is absent from the support $\mathtt{supp}(x) = \{b,d\}$, but also because swapping a with any other fresh atom $\mathbf{c}$ leaves the term unchanged: $(a\ \mathbf{c})\cdot \mathtt{f}(b,d) = \mathtt{f}(b,d)$. This invariance perspective allows us to decide freshness without knowing the exact support of x. It is enough to know that x has a finite support: if x remains unchanged under swaps of a with all but finitely many other atoms, then a cannot be in its support. This simplifies reasoning, because we no longer need to track precisely which atoms occur in x.

This inspired the work of Ayala et al. [1] who proposed replacing primitive freshness constraints $a\#X$ with primitive *fixed-point constraints* of the form $\pi\cdot X \approx_{\mathtt{C}} X$, abbreviated as $\pi \curlywedge_{\mathtt{C}} X$, where π is an arbitrary finite permutation. This approach directly captures the invariance view of freshness. However, a limitation of this approach is that when these constraints are not scoped under a $И$-quantifier, they do not fully capture the standard nominal semantics of freshness. For instance, even though $(a\ b)\cdot(a+b) =_{\mathtt{C}} a+b$ holds, neither a nor b is fresh for $a+b$.

To address this, the approach in [2] extends [1] by introducing nested $И$-quantification over fixed-point constraints of the form $И\overline{\mathbf{c}}.\pi \curlywedge_{\mathtt{C}} X$, where the list $\overline{\mathbf{c}}$ includes atoms that may occur in the permutation π. This design takes advantage of the fact that every finite permutation admits a canonical decomposition into disjoint cycles. Using this, any permutation π appearing in a constraint $И\overline{\mathbf{c}}.\pi \curlywedge_{\mathtt{C}} X$ can be split as $\pi = \pi_{\overline{\mathbf{c}}} \circ \pi_{\neg\overline{\mathbf{c}}}$, where $\pi_{\overline{\mathbf{c}}}$ includes all cycles that involve at least one atom from $\overline{\mathbf{c}}$, and $\pi_{\neg\overline{\mathbf{c}}}$ includes the remaining cycles. Since these cycles act on disjoint sets of atoms, the permutations $\pi_{\overline{\mathbf{c}}}$ and $\pi_{\neg\overline{\mathbf{c}}}$ have disjoint domains, making this factorization well-defined and unique. For example, in the constraint $И\mathbf{c_1},\mathbf{c_2}.(a\ \mathbf{c_1}\ d)\circ(e\ f)\circ(g\ \mathbf{c_2}) \curlywedge_{\mathtt{C}} X$, we have $\pi_{\overline{\mathbf{c}}} = (a\ \mathbf{c_1}\ d)\circ(g\ \mathbf{c_2})$ and $\pi_{\neg\overline{\mathbf{c}}} = (e\ f)$.

An important feature of this framework is that such constraints satisfy the equivalence: $И\overline{\mathbf{c}}.\pi \curlywedge_{\mathtt{C}} X$ if and only if $И\overline{\mathbf{c}}.(\pi_{\overline{\mathbf{c}}} \curlywedge_{\mathtt{C}} X \wedge \pi_{\neg\overline{\mathbf{c}}} \curlywedge_{\mathtt{C}} X)$. This shows that $И$-quantified fixed-point constraints extend primitive freshness: they can be viewed as a combination of freshness-like constraints (captured by $\pi_{\overline{\mathbf{c}}}$) and invariant, "pure" fixed-points (captured by $\pi_{\neg\overline{\mathbf{c}}}$).

It is worth noting that one practical advantage of fixed-point constraints is that because they can be seen as a special case of equality between nominal terms, the proof system can rely solely on equality rules. Unlike freshness constraints, we do not need a separate set of rules for relations like $a\#X$.

A limitation of previous works is that they have focused on specific equational theories such as commutativity (`C`), associativity (`A`), and associativity-commutativity (`AC`). In this talk, we take a further step by presenting a proof system for term equality modulo α-equivalence and an arbitrary equational theory $\mathtt{T}$, based on $И$-quantified permutation fixed-point constraints. This approach integrates ideas

from nominal algebra [5] and recent developments in reasoning modulo commutativity [2]. In this talk, we will outline these notions and discuss the main ideas and recent advances regarding the soundness and completeness of this system, providing a conceptual foundation for reasoning in a broader class of theories.

Going back to the framework of freshness constraints, we see that it provides not only a syntax and a proof system, but also a solid algebraic foundation. In fact, it extends some classical results from universal algebra, such as the HSP theorem (Homomorphisms, Subalgebras, Products). The nominal version of HSP [4] says that classes of nominal algebras that are closed under homomorphic images, subalgebras, products, and atom abstraction are exactly the ones that can be described by equations. This gives us a useful link between syntactic reasoning (equations with freshness) and semantic closure properties.

Freshness constraints in the original nominal algebra may initially seem outside the scope of equational reasoning. However, the nominal HSP theorem shows that they admit an algebraic characterization. This result allows the original system based on freshness to qualify as a universal algebra in the classical sense. In this talk, we investigate whether a similar characterization can be established for the extended system based on $\mathcal{U}$-quantified fixed-point constraints.

Our current findings indicate that the standard formulation of the HSP theorem fails for this extended setting. Intuitively, this happens because surjective homomorphisms between algebras can identify elements that are not equal in the domain, which may cause fixed-point conditions to hold in the codomain even when they do not in the domain. To address this issue, we propose to restrict attention to *rigid* homomorphisms, that is, those that preserve fixed-points, as a potential path to recovering a meaningful HSP-style result. The talk will outline this problem and key open questions.

References

[1] Mauricio Ayala-Rincón, Maribel Fernández & Daniele Nantes-Sobrinho (2020): *On Nominal Syntax and Permutation Fixed Points*. *Log. Methods Comput. Sci.* 16(1), p. 19:1–19:36, doi:10.23638/LMCS-16(1:19)2020.

[2] Ali K. Caires-Santos, Maribel Fernández & Daniele Nantes-Sobrinho (2025): *Equational Reasoning Modulo Commutativity in Languages with Binders*. In: *Proceedings of the 30th International Conference on Automated Deduction (CADE-30)*, Lecture Notes in Computer Science, Springer, pp. 632–652, doi:10.1007/978-3-031-99984-0_33. Extended version available at arXiv:2502.19287.

[3] Maribel Fernández & Murdoch James Gabbay (2007): *Nominal Rewriting*. *Inf. Comput.* 205(6), pp. 917–965, doi:10.1016/j.ic.2006.12.002.

[4] Murdoch James Gabbay (2009): *Nominal Algebra and the HSP Theorem*. *J. Log. Comput.* 19(2), pp. 341–367, doi:10.1093/logcom/exn055.

[5] Murdoch James Gabbay & Aad Mathijssen (2009): *Nominal (Universal) Algebra: Equational Logic with Names and Binding*. *J. Log. Comput.* 19(6), pp. 1455–1508, doi:10.1093/logcom/exp033.

[6] Murdoch James Gabbay & Andrew M. Pitts (2002): *A New Approach to Abstract Syntax with Variable Binding*. *Formal Aspects Comput.* 13(3-5), pp. 341–363, doi:10.1007/s001650200016.

[7] Christian Urban, Andrew M. Pitts & Murdoch James Gabbay (2004): *Nominal Unification*. *Theor. Comput. Sci.* 323(1-3), pp. 473–497, doi:10.1016/j.tcs.2004.06.016.

Termination Modulo Commutativity for Nominal Systems (Work in Progress)

Daniella Santaguida
University of Brasília, Brazil

Termination is an essential property for term rewrite systems and is necessary for completion procedures, which are widely studied and known in first-order. However, when working with languages with binders, one has to reason modulo alpha-equivalence and handle freshness conditions. In addition, these have to be integrated into rewriting derivations, adding challenges to the process. This challenge becomes greater when we add an equational theory to the reasoning, especially when we have axioms that may not be oriented without losing termination: such as Commutativity. This paper presents a generalisation of the recursive path ordering modulo α-equivalence and permutation equations (specifically the commutative one), in order to build a completion procedure that will work whenever one has confluence modulo C.

Overview

When studying term rewriting systems, one property that is often sought is termination. Even though termination is undecidable in general, we can show that a particular rewriting system terminates using reduction order. Basically, if we can show that $l > r$ for all system rules of the form $l \to r$, given some specific reduction order $>$, then the rewriting system is terminating. The only thing left to do is to find a specific order that works for the rewriting system in question. The real challenge is to find a reduction order that works with as many systems as possible.

One of the best-known reduction orders used in first-order term rewriting systems is Recursive Path Ordering (RPO) [3], which can be used in fully automated termination proofs [2]. Essentially, the idea of RPOs involves initially comparing the root symbols of two terms, followed by a recursive comparison of the collections of their immediate subterms. If we consider these collections as unordered multisets, we are dealing with the multiset path order; on the other hand, if we consider the collections as ordered tuples, we have the lexicographic path order. The RPO is nothing more than a combination of both.

Permutation equations are of the form $f(x_1,\ldots,x_n) \approx f(x_{\pi(1)},\ldots,x_{\pi(n)})$, where π is a permutation on $\{1,\ldots,n\}$ and f is a n-ary symbol [1]. Commutativity can be expressed using the permutation equation $f^C(x,y) \approx f^C(y,x)$, where f^C is a binary symbol. This equation cannot be oriented into a rewrite rule without losing termination of the entire system, which means the equation cannot be oriented by well-founded orderings. In order to circumvent this problem, a first-order version of the RPO working modulo permutation equations was given in [6]. The proposal was to use a reduction order $\succ_E$, for a set of permutation equations E, that is E-compatible and is useful for the termination of a rewriting system modulo E. We would like to do something similar, but for the nominal framework and considering only the commutativity as a permutation equation.

The nominal framework [5] has become a promising method for dealing with languages involving binders such as the lambda calculus and first-order logic. Within this framework, α-equivalence (denoted as $\approx_\alpha$) is used instead of syntactic equality, and freshness constraints are directly integrated within the nominal reasoning rather than being left to be handled by the meta-language. We then use expressions

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. x–xii, doi:10.4204/EPTCS.430.0.5

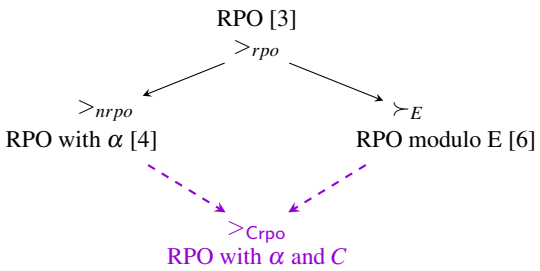


Figure 1: This work: nominal RPO with commutative equations

like $a\#X$ ("a is fresh for X") to say that the name a does not belong in the set of free variables of instances of the variable X. In other words, if a occurs in instances of X it must be abstracted by some binder, similarly to λ in the lambda calculus, or $\exists,\forall$-quantification in first-order logic. Note that a and X are variables at different levels: X is a meta-level *variable* that represents an entire expression and can be instantiated, but not bound by abstractions; whereas a is an object-level variable, called *atom*, which can be bound, but not instantiated.

Extending first-order rewriting to work modulo both α-equivalences and commutative axioms has its difficulties. First we need to distinguish the set of oriented equations – the set of rewrite rules R – and the set of unoriented ones – the commutative set C, which consists of a set built by commutative permutation equations. Then we will need two different rewrite relations: nominal R/C-rewriting applies rules from R to the equivalence class modulo $\approx_{\alpha,C}$ of a nominal term t, while R,C-rewriting uses nominal *C-matching* to determine if a rule in R applies to a nominal term, say t. This way, we take the commutativity into account without explicitly using it as a rule. The challenge is that the definition of the relations R/C and R,C also features freshness conditions, since this nominal term t may have variables.

In this talk, we will present our work in progress in the development of a reduction order for commutative nominal rewrite systems called $>_{\mathsf{Crpo}}$. This ordering brings together the ideas behind orderings dealing with α-equivalence from [4] and orderings dealing with permutation equations from [6], and in our case we consider specifically the commutative equation (see Figure 1). This is a work in progress and we still want to present a completion procedure for commutative nominal rewrite systems in the future.

References

[1] Jürgen Avenhaus (2004): *Efficient Algorithms for Computing Modulo Permutation Theories*. In David A. Basin & Michaël Rusinowitch, editors: *Automated Reasoning - Second International Joint Conference, IJCAR 2004, Cork, Ireland, July 4-8, 2004, Proceedings*, *Lecture Notes in Computer Science* 3097, Springer, pp. 415–429, doi:10.1007/978-3-540-25984-8_31.

[2] Franz Baader & Tobias Nipkow (1998): *Term rewriting and all that*. Cambridge University Press.

[3] Nachum Dershowitz (1982): *Orderings for term-rewriting systems*. *Theoretical Computer Science* 17(3), pp. 279–301, doi:10.1016/0304-3975(82)90026-3.

[4] Maribel Fernández & Albert Rubio (2012): *Nominal Completion for Rewrite Systems with Binders*. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts & Roger Wattenhofer, editors: *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, *Lecture Notes in Computer Science* 7392, Springer, pp. 201–213, doi:10.1007/978-3-642-31585-5_21.

[5] Murdoch Gabbay & Andrew M. Pitts (2002): *A New Approach to Abstract Syntax with Variable Binding*. *Formal Aspects Comput.* 13(3-5), pp. 341–363, doi:10.1007/s001650200016.

[6] Dohan Kim & Christopher Lynch (2021): *An RPO-Based Ordering Modulo Permutation Equations and Its Applications to Rewrite Systems*. In Naoki Kobayashi, editor: *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, *LIPIcs* 195, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 19:1–19:17, doi:10.4230/LIPICS.FSCD.2021.19.

A Function-Set Framework: General Properties and Applications to Modal Logic

Luke Bayzid Alexandre Madeira Manuel A. Martins
Mathematics Department of University of Aveiro
CIDMA - Research Center in Mathematics and Applications, University of Aveiro, Portugal
luke.adrian@ua.pt madeira@ua.pt martins@ua.pt

Representations are essential to mathematically model phenomena, but there are many options available. While each of those options provides useful properties with which to solve problems related to the phenomena in study, comparing results between these representations can be non-trivial, as different frameworks are used for different contexts. We present a general structure based on set-theoretic concepts that accommodates many situations related to logical and semantic frameworks. We show the versatility of this approach by presenting alternative constructions of modal logic; in particular, all modal logics can be represented within the framework.

1 Introduction

When attempting to model a phenomenon, we have access to a variety of different representations, many in completely distinct areas of mathematics, computer science, and beyond [3, 10]. This diversity of models is helpful because the particularity of each representation grants useful properties to study appropriate objects. However, objects represented in distinct areas can be hard to compare or program in a standardized form, especially if we wish to do so constructively.

The main motivation for this short paper, which is part of the first author's master's thesis [2], is to define a framework based on entities and time, applying it to modal logic thereafter. Our goal, as such, was to generalize what these concepts mean while still capturing broad properties. Since set theory is a well-studied area of mathematics with a wide reach, we chose it as the basis.

Before formally defining the system, we believe that it is important to slowly build up the intuition that led to the creation thereof. For this, we can start with a practical example and generalize it until we arrive at our system.

Let us suppose that we wish to represent a town of people who go to work during certain parts of the day. Already, we have made several assumptions — that there is a notion of time, actions, and people. These three concepts form the foundation of our system. In fact, any question about the described collection can be seen as a question about an entity's state at a point in time; if we wished to include the environment, it, too, could be considered an entity or merely something encoded within the states.

Now that we have extracted the objects of our system, we should formalize them and give them properties: Time is typically treated as the set of natural numbers, but the most important aspect is that it is totally ordered. States encode the information that we wish to study, and as that can vary significantly, we cannot particularize them more than just to say that they form a set. Since the semantics of the system are encoded within the state, the only important part about entities is that they are distinguishable; as such, we only need to store a set of cardinals — an indexing system, much like a computer's memory addresses. It should be noted that even if we do not care about encoding time, we can make it a singleton.

It should now be clear that we are a approaching a temporal, entity-based indexing system that points to the semantics of an object at a particular moment. This is not the only generalization of the above, but

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. 1–7, doi:10.4204/EPTCS.430.1

it is the one that we followed. With these intuitions set, we will now formally define the above system in Section 2, use the definition to describe a common property in Section 2.1, and apply the framework to modal logic in Section 3, where we prove that modal logic can, in a sense, be seen as a particularization thereof in Theorem 3.1. We provide relevant future work to show the framework's direction in Section 4.

2 Framework

Let $S \neq \emptyset$ be a set of states, $E \neq \emptyset$ be a set of entities, and $T \neq \emptyset$ be a totally ordered set. Then, we call $\Omega \subseteq S^{E\times T}$ a *context*, which is the primary object of study; an element of a context is called an *instance*. Intuitively, each instance represents a possible timeline, where its corresponding context represents the set of all possible timelines. As is commonly known, we also have that $S^{E\times T} \cong S^{E^T} \cong S^{T^E}$, which will be useful later. In this context, given a $\omega \in S^{E\times T}$ (i.e., a function $\omega : E \times T \to S$), we write

- ω^* for the corresponding element of S^{E^T} (i.e., the function $\omega^* : T \to (E \to S)$) and
- $\omega^@$ for the corresponding element of S^{T^E} (i.e., the function $\omega^@ : E \to (T \to S)$),

whereas a subscript of the same type is used to take an element of S^{E^T} or S^{T^E} and turn it into one of $S^{E\times T}$, such as ω_* or $\omega_@$.

Example 1. *Let us consider a trivial example to illustrate the notation — Alice and Bob live together, and it is known that when Bob is at home, Alice will be at home an hour later. We can define the aforementioned situation as the following:*

$$S = \{\text{Home}, \text{Out}\};$$
$$E = \{\text{Alice}, \text{Bob}\};$$
$$T = \mathbb{N}; \text{ and}$$
$$\Omega = \{\omega \in S^{E\times T} \mid \forall t \in T, \omega(\text{Bob}, t) = \text{Home} \Rightarrow \omega(\text{Alice}, t+1) = \text{Home}\}.$$

If we then wished to add that Bob will be at home during odd hours, we could write the following:

$$\Omega' = \{\omega \in \Omega \mid \forall t \in T, t \bmod 2 = 1 \Rightarrow \omega(\text{Bob}, t) = \text{Home}\}.$$

Definition 1. *When analyzing an instance up to a particular point, it is typically useful to know its neighborhood — similar possibilities permitted by the context. For this, we use the* consistency context *of $\tilde{\omega}$ up to t, which is defined as follows:*

$$\mathrm{C}^{\tilde{\omega},t}(\Omega) = \{\omega \in \Omega \mid \forall e \in E, \forall t' \in T, t' \leq t \Rightarrow \omega(e,t') = \tilde{\omega}(e,t')\}.$$

Intuitively, the above definition captures all of the instances that agree with $\tilde{\omega}$ up to t; a similar function can be made if t is meant to be excluded.

2.1 Determinability

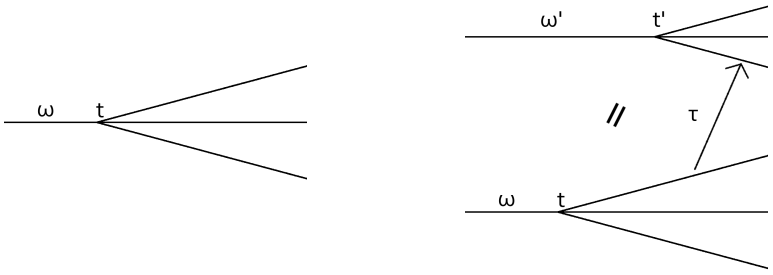
Determinism is an important concept in philosophy [1], computer science [7], and physics [11], as it refers to the determination of the future by using past information. This can be seen as a subset of determining all specific futures that are consistent with past information, which is usually called indeterminism.

Definition 2. *We model both concepts with contexts under the term* determinability*, which is done using the following:*

$$\forall \omega, \omega' \in \Omega, \forall t, t' \in T, \exists \tau \in \mu(t^+, t'^+), \tag{1}$$
$$\omega^*(t) = \omega'^*(t') \Rightarrow \beta(\tau) \wedge \{\tilde{\omega}^*|_{t^+} \mid \tilde{\omega} \in \mathrm{C}^{\omega,t}(\Omega)\} = \{\tilde{\omega}^*|_{t'^+} \circ \tau \mid \tilde{\omega} \in \mathrm{C}^{\omega',t'}(\Omega)\}, \tag{2}$$

where $t^+ = \{t' \in T \mid t' \geq t\}$, where $\mu(A,B)$ is the set of all monotonous functions from A to B, and where $\beta(\tau)$ qualifies τ as a bijection.

Here, the idea is to say that if two instances have the same state at certain moments, then the futures generated by them are the same (i.e., consistent). In particular, if there is only one future at each point (possibly excluding the minimum of T if it exists), we say that the context is deterministic.



(a) Consistency Context (b) Determinability

Example 2. *Let us consider* Chess $\subseteq ((\mathbf{8}^2 \cup \mathbf{1})^{16})^{\mathbf{2}\times\mathbb{N}}$. *The idea is to encode chess as a function that keeps track of the pieces (*16*) of each player (*$\mathbf{2}$*), marking them on the board (*$\mathbf{8}^2$*) or outside of it (*$\mathbf{1}$*) over* $\mathbb{N}$. *With this encoding, chess is not determinable. One reason is that we cannot know who is next by looking at the state of the board. To amend that, we could include another entity that keeps track of the turns; alternatively, we could extend the states, such as* $((\mathbf{8}^2 \cup \mathbf{1})^{16} \times \mathbf{2})^{\mathbf{2}\times\mathbb{N}}$, *such that a player can only make their turn if the second coordinate (the added portion) is one, which would alternate as turns progress. Beyond this, certain rules of chess might require similar changes to grant determinability.*

Determinability With Countable Time. In contexts with countable time, we can actually use a slightly different but intuitive understanding of determinability. In a sense, we expect any countable, determinable system to be describable by a function that iterates the present into the future.

Definition 3. *Let $A = \{\omega^*(t) \mid (t,\omega) \in T \times \Omega\}$. We say that Ω has an iterator whenever $\exists i \in \mathrm{P}(A)^A, \forall \omega \in \Omega, \forall t \in T, (C^{\omega,t}(\Omega))^*(t+1) = i(\omega^*(t))$.*

Theorem 2.1. *Let T be countable. Let $\Omega \subseteq S^{E\times T}$ be determinable. Then, Ω has an iterator.*

Proof. Let us say that i is a binary relation of the form $A \times P(A)$ such that

$$(a,b) \in i \Rightarrow a \in \Omega^*(t) \wedge b = (C^{\omega,t}(\Omega))^*(t+1)$$

where $\omega \in \Omega$ such that $\omega^*(t) = a$ for some $t \in T$. We will show that i really is a function, for which we must prove that if $a = a'$ in $(a,b) \in i \wedge (a',b') \in i$, then $b = b'$. Since $a = a'$, we have that $\exists \omega, \omega' \in \Omega, \exists t,t' \in T, \omega^*(t) = a = a' = \omega'^*(t')$. Given determinability, we know that $\exists \tau \in \mu(t^+,t'^+)$,

$$\beta(\tau) \wedge \{\tilde{\omega}^*|_{t^+} \mid \tilde{\omega} \in \mathrm{C}^{\omega,t}(\Omega)\} = \{\tilde{\omega}^*|_{t'^+} \circ \tau \mid \tilde{\omega} \in \mathrm{C}^{\omega',t'}(\Omega)\}.$$

By bijective monotony over a total order, since t is the minimum of t^+, it must also get mapped by τ to the minimum of t'^+, which is t', for which the same applies to $t+1$ and $t'+1$ by virtue of bijectivity and countability. As such, we can particularize the above to

$$\{\tilde{\omega}^*(t+1) \mid \tilde{\omega} \in \mathrm{C}^{\omega,t}(\Omega)\} = \{\tilde{\omega}^*(t'+1) \mid \tilde{\omega} \in \mathrm{C}^{\omega',t'}(\Omega)\},$$

which is to say that $(C^{\omega,t}(\Omega))^*(t+1) = (C^{\omega',t'}(\Omega))^*(t'+1)$. Therefore, $b = b'$, meaning that i truly is a function. □

What is slightly harder to prove is that the converse is true, meaning that having an iterator is equivalent to being determinable in contexts with countable time, which confirms our intuitions.

Theorem 2.2. *Let T be countable. Let $\Omega \subseteq S^{E\times T}$ have an iterator. Then, Ω is determinable.*

Proof. Let us assume that Ω has i. Let us also assume that $\omega^*(t) = \omega'^*(t')$. Since i exists, we have that

$$i(\omega^*(t)) = (C^{\omega,t}(\Omega))^*(t+1) = (C^{\omega',t'}(\Omega))^*(t'+1) = i(\omega'^*(t')).$$

This means that

$$\{\tilde{\omega}^*(t+1) \mid \tilde{\omega} \in \mathrm{C}^{\omega,t}(\Omega)\} = \{\tilde{\omega}^*(t'+1) \mid \tilde{\omega} \in \mathrm{C}^{\omega',t'}(\Omega)\}.$$

This both proves the result for the base case of t^+ and t'^+ and can be inductively applied to the remaining elements thereof (i.e., use $i(\tilde{\omega}^*(t))$ to prove the result for $t+1$, inductively proving the result for t^+ for all $t \in T$). With $\cdot + 1$, we create a monotonous bijection from t^+ to t'^+: $\forall n \in \mathbb{N}_1, \tau : t + n \mapsto t' + n$. □

3 Modal Context

In this section, we will focus on modeling standard modal logic [4] using contexts and show that this framework is a generalization thereof, allowing us to capture the internal aspects of a Kripke frame. We call our constructions *modal contexts* because we treat each instance as a modal world, where standard modal concepts are used. Although this section focuses on standard modal logic, the use of contexts [2] also reaches dynamic logic [8] and epistemic logic [6].

Modal Operators. From an abstract perspective, the point of using a context is to treat the states as black boxes. For this, we have to generalize the notion of applying a modal operator. We do this by creating a *modal operator*, which is an injection $\Box : S \to S$ such that

$$\forall s \in S, \forall n \in \mathbb{N}, \Box^{n+2}(s) \neq \Box(s),$$

which is to say that there are no loops (e.g., $\Box(\Box s) \neq \Box s$). Assuming $S \neq \emptyset$, this naturally requires $|S| \geq \aleph_0$. This is done so that we can encode that there is a syntactic distinction between $\Box\phi$ and ϕ.

Modal Context.

Definition 4. *Let us consider a power context $M \subseteq \mathbb{W} = \mathrm{P}(S)^{E\times T}$. Let $\Box$ and $\Diamond$ be modal operators. Let R be a binary relation on $\mathbb{W}$. We say that M is a modal context whenever $\forall \omega \in \mathbb{W}, \omega \in M \iff$*

$$\begin{gathered}((\forall e \in E, \forall t \in T, \forall s \in S, (\Box s \in \omega(e,t) \iff (\forall \omega' \in \omega R \cdot, s \in \omega'(e,t))))\\ \wedge\\ (\forall e \in E, \forall t \in T, \forall s \in S, (\Diamond s \in \omega(e,t) \iff (\exists \omega' \in \omega R \cdot, s \in \omega'(e,t))))).\end{gathered}$$

The primary idea here is that we apply modal operators to each of the states whenever related worlds share that underlying state. As we would hope, traditional modal logic can be modeled within this. To prove this, we relate each world (modal logic) to an instance. In particular, we say that $w \in W$ relates to $w' \in M$ whenever

$$\forall \phi \in \Phi, (M, w \models \phi) \iff \phi \in w'(0,0).$$

This naturally defines how we can prove statements in a modal context — To prove ϕ, we merely need to show that it belongs to a specific set.

Theorem 3.1. *Let $M = \langle W, R, V \rangle$ be a modal logic. Then, there is a modal context $M' \subseteq \mathbb{W} = \mathrm{P}(\Phi)^{\mathbb{1}\times\mathbb{1}}$ such that*

$$\forall w \in W, \exists w' \in M', \forall \phi \in \Phi, (M, w \models \phi) \iff \phi \in w'(0,0).$$

Proof. For this proof, we will focus on constructing M' and show that it is a modal context. We will first create an equivalence class over W. We do this by stating that

$$wQw' \iff (\forall \phi \in \Phi, (M, w \models \phi) \iff (M, w' \models \phi)),$$

where reflexivity, symmetry, and transitivity are all trivial to prove. We can now define M' by stating that $\forall \omega \in \mathbb{W}$,

$$\omega \in M' \iff (\exists [\omega'] \in W/_Q, \omega(0,0) = \{\phi \in \Phi \mid M, \omega' \models \phi\}).$$

Since each function is fully defined by the set to which it maps $(0,0)$ (all of whom are equal by class), the functions are in bijection with $W/_Q$. As such, we will denote the class of $W/_Q$ that is associated with ω by $[\omega]$.

To finish our preliminary constructions, we need to make create R', which we define as

$$\forall \omega, \omega' \in M', \omega R' \omega' \iff (\exists a \in [\omega], \exists b \in [\omega'], aRb).$$

We now just need to prove that M' is a modal context. We will start with the proof of $\Box$, which is to say that

$$\forall \omega \in M', \Box\phi \in \omega(0,0) \iff (\forall \omega' \in \omega R'\cdot, \phi \in \omega'(0,0)).$$

We will now prove $\Rightarrow$, meaning that we will assume that $\Box\phi \in \omega(0,0)$: Let $\omega \in M'$. If $\Box\phi \in \omega(0,0)$, we have that

$$\forall \omega' \in [\omega], (M, \omega' \models \Box\phi).$$

Then, by the properties of modal logic, we have that

$$\forall \omega' \in [\omega], \forall \omega'' \in \omega' R\cdot, (M, \omega'' \models \phi).$$

Let $\omega'' \in \omega R'\cdot$, which means that $\exists \omega' \in [\omega], \exists \omega''' \in [\omega''], \omega' R \omega'''$. However, with the above, that means that $M, \omega''' \models \phi$, which means that $\phi \in \omega''(0,0)$, concluding our proof of $\Rightarrow$.

We will now prove $\Leftarrow$, meaning that we will assume that $\forall \omega' \in \omega R'\cdot, \phi \in \omega'(0,0)$: Let $\omega' \in [\omega]$. Let $\omega'' \in \omega' R\cdot$. As such, we know that $\exists \omega''' \in W/_Q, \omega'' \in [\omega''']$ such that $\omega R' \omega'''$. Therefore, using our assumption, $\phi \in \omega'''(0,0)$, meaning that $\forall \omega'''' \in [\omega'''], (M, \omega'''' \models \phi)$ (in particular, ω''). Since this was done generically for $\omega'' \in \omega' R\cdot$, we have that $\forall \omega'' \in \omega' R\cdot, (M, \omega'' \models \phi)$, which means that $M, \omega' \models \Box\phi$ by the properties of modal logic. As such, $\Box\phi \in \omega(0,0)$, concluding our proof of $\Leftarrow$.

We now need to prove $\Diamond$, which is to say that

$$\forall \omega \in M', \Diamond\phi \in \omega(0,0) \iff (\exists \omega' \in \omega R'\cdot, \phi \in \omega'(0,0)).$$

Similarly to what was done above, we will divide this proof into two parts: $\Rightarrow$ and $\Leftarrow$.

We will now prove $\Rightarrow$, meaning that we will assume that $\Diamond\phi \in \omega(0,0)$: Let $\omega \in M'$. If $\Diamond\phi \in \omega(0,0)$, we have that

$$\forall \omega' \in [\omega], (M, \omega' \models \Diamond\phi).$$

Then, by the properties of modal logic, we have that

$$\forall \omega' \in [\omega], \exists \omega'' \in \omega' R \cdot, (M, \omega'' \models \phi).$$

Let us choose $\omega' \in [\omega]$. We now have the above ω'', whose class in $W/_Q$ we will call $[\omega''']$; in particular, we can say that $\omega''' \in M'$ because $W/_Q$ is in bijection with M'. $\omega''' \in \omega R' \cdot$ because $\omega' R \omega''$. Since $M, \omega'' \models \phi$, we have that $\phi \in \omega'''(0,0)$, which proves that $\exists \omega''' \in \omega R' \cdot, \phi \in \omega'''(0,0)$, concluding our proof of $\Rightarrow$.

We will now prove $\Leftarrow$, meaning that we will assume that $\exists \omega' \in \omega R' \cdot, \phi \in \omega'(0,0)$: If $\omega' \in \omega R' \cdot$, then $\exists a \in [\omega], \exists b \in [\omega'], aRb$, and since $b \in [\omega']$, we have that $M, b \models \phi$, which means that $M, a \models \Diamond \phi$. As such, $\Diamond \phi \in \omega(0,0)$, concluding our proof of $\Leftarrow$.

At last, we have proven that M' is a modal context whose instances capture the worlds of M. □

An interesting note about the above theorem is that we can faithfully capture modal logic without using labels for worlds. Moreover, this approach generically works for any power context.

4 Conclusion and Future Work

Contexts allow us to create by restriction — We choose a function set that is large enough and choose a subset to fit the situation. This perspective is in contrast to many standard approaches, such as a Kripke frame, where we might have to define the underlying structure and carefully add rules or components. This approach might not be adequate for all situations, but it may allow for other views on existing phenomena.

We have shown that the framework can be used to abstractly model structures without worrying about many details (such as what was done with chess and determinability). We have also shown that we can encode standard modal logic into a particularization of the framework, with Theorem 3.1 to constructively convert the standard axioms of modal logic.

However, this implementation of modal logic does not use E and T, which are primary points of interest for contexts; in fact, this practically just makes the context a set. This is because it is generally doubtful what E and T would mean unless we are discussing epistemic or temporal logic. We would like to investigate extensions of this definition that are also consistent with E and T, which, in the case of epistemic logic, is to say that the instance mapping of $e \in E$ should correspond to, for example, its knowledge as a power set, such as

$$\omega(\text{Alice}, t) = \{\text{"Bob bought a house."}, \text{"Bob knows that Alice knows that Bob bought a house."}, \dots\}.$$

In [2], we have an alternative definition to solve this for epistemic logic, though it is rather independent of how modal contexts were defined here. Temporal logic will also be approached, as the properties of T being a total order are typically sufficient for many common temporal systems.

Much of what has been presented here forms the basics of this framework, and there are many unanswered questions that we would like to explore in future work. This includes the following:

- Study function sets with broader kinds of domains, not just E and T. In particular, studying how the properties of the domain (such as T being a total order) affect the resulting context is the main interest.
- Model epistemic [6]/doxastic [9]/temporal [5] /dynamic [8] logic in a way that is coherent with E and T, meaning that each $e \in E$ and $t \in T$ should correspond to their modal counterparts. This may possibly require other constructions of modal contexts.

- Define bisimulations and modal invariance for modal contexts.
- Find the conditions under which expected properties (e.g., finite model property and decidability) still hold and techniques (e.g., filtration) can be applied.

Acknowledgments. This work is supported by FCT – Fundação para a Ciência e a Tecnologia through projects UIDB/04106/2025 at CIDMA and by National and European Funds through SACCCT- IC&DT - Sistema de Apoio à Criação de Conhecimento Científico e Tecnológico, as part of COMPETE2030, within the project BANSKY with reference number 15253.

References

[1] Roy F. Baumeister, Cory J. Clark & Stephan Lau (2022): *Determinism*, pp. 375–380. Springer International Publishing, Cham, doi:10.1007/978-3-030-90913-0_232.

[2] Luke Bayzid (2025): *A Specification Framework Using Function Sets: General Properties & Applications to Modal Logic*. Master's thesis, University of Aveiro.

[3] Dines Bjorner & Martin C. Henson (2007): *Logics of Specification Languages (Monographs in Theoretical Computer Science. An EATCS Series)*, 1st edition. doi:10.1007/978-3-540-74107-7.

[4] Patrick Blackburn, Maarten de Rijke & Yde Venema (2001): *Modal Logic*. Cambridge University Press, doi:10.1017/cbo9781107050884.

[5] Stéphane Demri, Valentin Goranko & Martin Lange (2016): *Temporal Logics in Computer Science: Finite-State Systems*, p. 716–736. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, doi:10.1017/CBO9781139236119.

[6] Hans van Ditmarsch, Wiebe van der Hoek & Barteld Kooi (2007): *Dynamic Epistemic Logic*. Springer, Dordrecht, Netherland, doi:10.1007/978-1-4020-5839-4.

[7] Garland, Joshua and James, Ryan and Bradley, Elizabeth (2013): *Determinism, Complexity, and Predictability in Computer Performance*. doi:10.48550/arXiv.1305.5408.

[8] David Harel, Dexter Kozen & Jerzy Tiuryn (2000): *Dynamic Logic*. MIT Press, doi:10.7551/mitpress/2516.001.0001.

[9] Sten Lindström & Wlodek Rabinowicz (1999): *DDL Unlimited: Dynamic Doxastic Logic for Introspective Agents*. *Erkenntnis* 50(2-3), pp. 353–385, doi:10.1023/a:1005577906029.

[10] John Michael Spivey (1989): *Understanding Z - a specification language and its formal semantics (reprint)*. *Cambridge tracts in theoretical computer science* 3, Cambridge University Press. Available at `https://www.cambridge.org/us/universitypress/subjects/computer-science/programming-languages-and-applied-logic/understanding-z-specification-language-and-its-formal-semantics`.

[11] Marij van Strien (2021): *Was physics ever deterministic? The historical basis of determinism and the image of classical physics*. *The European Physical Journal H* 46(1), p. 8, doi:10.1140/epjh/s13129-021-00012-x.

Sequent Calculi for Data-Aware Modal Logics

Carlos Areces
Universidad Nacional de Córdoba
and CONICET, Argentina
carlos.areces@unc.edu.ar

Valentin Cassano
Universidad Nacional de Río Cuarto
and CONICET, Argentina
valentin@dc.exa.unrc.edu.ar

Danae Dutto
Universidad Nacional de Córdoba
and CONICET, Argentina
ddutto@dc.exa.unrc.edu.ar

Raul Fervari
Universidad Nacional de Córdoba
and CONICET, Argentina
rfervari@unc.edu.ar

Data-aware modal logics offer a powerful formalism for reasoning about semi-structured queries in languages such as DataGL, XPath, and GQL. In brief, these logics can be viewed as modal systems capable of expressing both reachability statements and data-aware properties, such as value comparisons. One particularly expressive logic in this landscape is $\mathrm{HXPath}_{\mathrm{D}}$, a hybrid modal logic that captures not only the navigational core of XPath but also data comparisons, node labels (keys), and key-based navigation operators. While previous work on $\mathrm{HXPath}_{\mathrm{D}}$ has primarily focused on its model-theoretic properties, in this paper we approach $\mathrm{HXPath}_{\mathrm{D}}$ from a proof-theoretic perspective. Concretely, we present a sound and complete Gentzen-style sequent calculus for $\mathrm{HXPath}_{\mathrm{D}}$. Moreover, we show all rules in this calculus are invertible, and that it enjoys cut elimination. Our work contributes to the proof-theoretic foundations of data-aware modal logics, and enables a deeper logical analysis of query languages over graph-structured data. Moreover, our results lay the groundwork for extending proof-theoretic techniques to a broader class of modal systems.

1 Introduction

Semi-structured data organization [18] is reemerging as a flexible paradigm for representing and storing information, offering a contrast to the rigid rows-and-columns model of traditional relational databases. This approach is exemplified by *graph databases*, where information is represented and stored in a semi-structured form as *data graphs*, in which nodes and edges are labeled with values that capture both structure and data content [36]. Graph databases are widely adopted in domains such as the web, social networks, fraud detection, and recommendation engines, and are implemented in tools like Neo4j and Amazon Neptune. To support such range of applications, graph databases rely on expressive query languages designed to navigate and extract information from their underlying structures. However, unlike relational databases, where SQL serves as the *de facto* standard, query languages for graph databases are still evolving. Recent efforts, as discussed in [27, 28], are converging toward the Graph Query Language (GQL) Standard [30], which aims to unify existing industrial approaches. GQL builds on its academic predecessor G-CORE [7] and draws conceptual inspiration from Regular Path Queries [32] and the XML Path Language (XPath) [31], offering expressive capabilities for navigating graph structures and performing equality and inequality tests on data. This connection highlights the importance of logical foundations in understanding and advancing modern query languages for graph data.

In this article, we focus on XPath, approaching it from a logical perspective and building on prior work that formalizes fragments of this language as modal logics. Below, we outline a brief timeline highlighting key developments in this direction.

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. 8–25, doi:10.4204/EPTCS.430.2

Initial foundational results [15, 20], showed that the navigational fragment of XPath—known as Core-XPath [29]—is strictly less expressive than Propositional Dynamic Logic (PDL) over trees. This set the stage for subsequent work, ending with a complete axiomatization for Core-XPath in [20, 19]. As the study of XPath progressed, attention turned to features that go beyond pure navigation. Querying data graphs often requires the ability to reason about and compare data values—a limitation of purely navigational logics. How to properly represent and manipulate data within a modal setting gave rise to the development of *data-aware modal logics*: extensions of modal logic that incorporate operators for handling data alongside navigation. These logics are tailored to capturing the interplay between structure and data in graph-based systems, integrating mechanisms such as data comparison, binding, and quantification within the modal framework. This includes the development of expressive yet decidable languages, the design of automata-theoretic characterizations, and the identification of suitable semantics that balance expressive power with computational tractability (see e.g., [21]). The result is a family of expressive formalisms capable of modeling sophisticated queries over data-rich systems.

These efforts have opened up new avenues in both theoretical studies and practical applications, particularly in areas such as verification, knowledge representation, and query languages for graph databases. For instance, [16] introduces Core-Data-XPath as an extension of Core-XPath with equality and inequality comparisons. The satisfiability problem and complexity aspects of fragments of these logics are explored in [26, 22, 31, 23], while the model theory and bisimulation algorithms for this kind of logics are studied in [4, 24, 25, 1, 5, 2]. Axiomatic systems extending those from [19] are presented in [3, 6]. This theoretical groundwork led to concrete applications. For example, [29] discusses newly proposed algorithms with existing XPath processors, while the satisfiability methods in [23] can be used to check consistency in XML documents. Furthermore, the bisimulation algorithms in [25, 1] have applications in database minimization, while [13] examines how the complexity of fragments impacts real-world applications.

The work in [10] introduces a novel extension to data-aware modal logics by incorporating *keys*—or labels—for nodes, and *jump-to-key* operations. This results in the logic we refer to as $\mathrm{HXPath_D}$, which extends Core-Data-XPath with expressive constructs from Hybrid Logic. In particular, nominals and the satisfiability operator @, well-studied in Hybrid Logics [9], provide the means to refer directly to specific nodes, and to "navigate" to specific nodes. This addition enables $\mathrm{HXPath_D}$ to express properties and navigation patterns not previously captured in earlier formalisms. Furthermore, the hybrid features of $\mathrm{HXPath_D}$ support a novel axiomatization, with completeness established using Henkin-style models.

In this article, we contribute to the logical study of data-aware modal logics from a *proof-theoretic* perspective by developing a sequent calculus for $\mathrm{HXPath_D}$. We establish the completeness of our calculus by leveraging the completeness results in [10]. We also show that our calculus enjoys cut-elimination and invertibility of rules. Furthermore, we relate our work to research on labeled modal sequent calculi with equality [35] and the proof theory of hybrid logic [17]. Developing a proof theory for modal logics is notoriously difficult—see, e.g., the discussion in [34]. Several techniques, such as display calculi [38], hypersequents [12], deep inference [37], labeled systems [33], and hybridization [17], have been explored to establish a solid proof-theoretic foundation for various modal logics. Our results suggest that labeling and hybridization can also support a well-grounded proof theory for data-aware modal logics.

Outline. Sec. 2 introduces the logic $\mathrm{HXPath_D}$. Sec. 3 presents the sequent calculus **G** for $\mathrm{HXPath_D}$. Key properties of **G**—soundness, completeness, and invertibility of rules—are addressed in Secs. 3.1 to 3.3. Cut elimination is established in Sec. 4. Sec. 5 demonstrates a correspondence between **G** and the sequent calculus for Basic Hybrid Logic from [17]. Sec. 6 concludes with final remarks and directions for future work. Details and omitted proofs are available in [11].

2 Hybrid XPath with Data

We begin by presenting the formal syntax and semantics of $\mathrm{HXPath}_{\mathsf{D}}$—further details are found in [10]. In our presentation, we assume Prop, Nom, Mod, and Cmp be pairwise disjoint sets of symbols for propositions, nominals, modalities, and data comparisons, respectively. Moreover, we assume that Mod and Cmp are finite, while Prop and Nom are countably infinite.

Definition 1. *The language of* $\mathrm{HXPath}_{\mathsf{D}}$ *has* path *expressions (denoted* $\alpha, \beta, \ldots$*) and* node *expressions (denoted* $\varphi, \psi, \ldots$*), mutually defined by the grammar:*

$$\alpha, \beta \coloneqq \mathsf{a} \mid i{:} \mid \varphi? \mid \alpha\beta$$
$$\varphi, \psi \coloneqq p \mid i \mid \bot \mid \varphi \to \psi \mid @_i\varphi \mid \langle \mathsf{a} \rangle \varphi \mid \langle \alpha =_{\mathsf{c}} \beta \rangle \mid \langle \alpha \neq_{\mathsf{c}} \beta \rangle,$$

where $p \in \mathsf{Prop}$, $i \in \mathsf{Nom}$, $\mathsf{a} \in \mathsf{Mod}$, *and* $\mathsf{c} \in \mathsf{Cmp}$.[1] *For path expressions, we use* $\varepsilon \coloneqq \top?$ *to indicate the* empty *path. For node expressions, we use standard abbreviations:* $\top \coloneqq \bot \to \bot$, $\neg\varphi \coloneqq \varphi \to \bot$, $\varphi \vee \psi \coloneqq \neg\varphi \to \psi$, $\varphi \wedge \psi \coloneqq \neg(\varphi \to \neg\psi)$, *and* $\varphi \leftrightarrow \psi \coloneqq (\varphi \to \psi) \wedge (\psi \to \varphi)$. *We also abbreviate:* $\langle j{:} \rangle \varphi \coloneqq @_j\varphi$, $\langle \psi? \rangle \varphi \coloneqq \psi \wedge \varphi$, $\langle \alpha\beta \rangle \varphi \coloneqq \langle \alpha \rangle \langle \beta \rangle \varphi$, *and* $[\alpha]\varphi \coloneqq \neg\langle \alpha \rangle \neg\varphi$. *These abbreviations complete all cases to allow arbitrary paths* α *inside a unary modality* $\langle \alpha \rangle$. *Finally, we abbreviate* $[\alpha \blacktriangle \beta] \coloneqq \neg\langle \alpha \blacktriangledown \beta \rangle$. *In the last abbreviation, we use* $\blacktriangle$ *when there is no need to distinguish* $=_{\mathsf{c}}$ *and* $\neq_{\mathsf{c}}$, *and use* $\blacktriangledown$ *to indicate* $\neq_{\mathsf{c}}$ *if* $\blacktriangle$ *is* $=_{\mathsf{c}}$, *and to indicate* $=_{\mathsf{c}}$ *if* $\blacktriangle$ *is* $\neq_{\mathsf{c}}$.

Path and node expressions are interpreted over hybrid data models.

Definition 2. *A* (hybrid data) model *is a tuple* $\mathfrak{M} = \langle \mathrm{N}, \{\mathrm{R}_{\mathsf{a}}\}_{\mathsf{a} \in \mathsf{Mod}}, \{\approx_{\mathsf{c}}\}_{\mathsf{c} \in \mathsf{Cmp}}, g, \mathrm{V} \rangle$, *where* N *is a non-empty set of nodes; each* R_{a} *is a (binary)* accessibility relation *on* N*; each* $\approx_{\mathsf{c}}$ *is an equivalence relation on* N, *called a* comparison*;* $g : \mathsf{Nom} \to \mathrm{N}$ *is a* nominal assignment*; and* $\mathrm{V} : \mathsf{Prop} \to 2^{\mathrm{N}}$ *is a* valuation.

The satisfiability relation for path and node expressions is as follows.

Definition 3. *Let* $\mathfrak{M} = \langle \mathrm{N}, \{\mathrm{R}_{\mathsf{a}}\}_{\mathsf{a} \in \mathsf{Mod}}, \{\approx_{\mathsf{c}}\}_{\mathsf{c} \in \mathsf{Cmp}}, g, \mathrm{V} \rangle$ *be a model, and let* $\{n, n'\} \subseteq \mathrm{N}$. *The satisfiability relation* $\Vdash$ *is given by the following conditions:*

$\mathfrak{M}, n, n' \Vdash \mathsf{a}$	*iff* $n \mathrm{R}_{\mathsf{a}} n'$
$\mathfrak{M}, n, n' \Vdash i{:}$	*iff* $g(i) = n'$
$\mathfrak{M}, n, n' \Vdash \varphi?$	*iff* $n = n'$ *and* $\mathfrak{M}, n \Vdash \varphi$
$\mathfrak{M}, n, n' \Vdash \alpha\beta$	*iff exists* $n'' \in \mathrm{N}$ *s.t.* $\mathfrak{M}, n, n'' \Vdash \alpha$ *and* $\mathfrak{M}, n'', n' \Vdash \beta$
$\mathfrak{M}, n \Vdash p$	*iff* $n \in \mathrm{V}(p)$
$\mathfrak{M}, n \Vdash i$	*iff* $g(i) = n$
$\mathfrak{M}, n \Vdash \bot$	*never*
$\mathfrak{M}, n \Vdash \varphi \to \psi$	*iff* $\mathfrak{M}, n \Vdash \varphi$ *implies* $\mathfrak{M}, n \Vdash \psi$
$\mathfrak{M}, n \Vdash @_i\varphi$	*iff* $\mathfrak{M}, g(i) \Vdash \varphi$
$\mathfrak{M}, n \Vdash \langle \mathsf{a} \rangle \varphi$	*iff exists* $n' \in \mathrm{N}$ *s.t.* $\mathfrak{M}, n, n' \Vdash \mathsf{a}$ *and* $\mathfrak{M}, n' \Vdash \varphi$
$\mathfrak{M}, n \Vdash \langle \alpha =_{\mathsf{c}} \beta \rangle$	*iff exists* $n', n'' \in \mathrm{N}$ *s.t.* $\mathfrak{M}, n, n' \Vdash \alpha$, $\mathfrak{M}, n, n'' \Vdash \beta$ *and* $n' \approx_{\mathsf{c}} n''$
$\mathfrak{M}, n \Vdash \langle \alpha \neq_{\mathsf{c}} \beta \rangle$	*iff exists* $n', n'' \in \mathrm{N}$ *s.t.* $\mathfrak{M}, n, n' \Vdash \alpha$, $\mathfrak{M}, n, n'' \Vdash \beta$ *and* $n' \not\approx_{\mathsf{c}} n''$.

Let Ψ *be a set of node expressions, we use* $\mathfrak{M}, n \Vdash \Psi$ *to indicate* $\mathfrak{M}, n \Vdash \psi$ *for all* $\psi \in \Psi$. *We say* Ψ *is* satisfiable *iff there exists* $\mathfrak{M}, n$ *s.t.* $\mathfrak{M}, n \Vdash \Psi$. *We call a node expression* φ *a* consequence *of* Ψ, *written* $\Psi \vDash \varphi$, *iff* $\Psi \cup \{\neg\varphi\}$ *is unsatisfiable. If* Ψ *is the empty set, we write* $\vDash \varphi$ *and call* φ *a* tautology.

Proposition 4, which follows directly from Def. 3, confirms that the abbreviations behave as intended.

[1] Unlike [10], we treat $@_i\varphi$ and $\langle \mathsf{a} \rangle \varphi$ as primitive. This choice simplifies the formulation of the sequent calculus in Sec. 3.

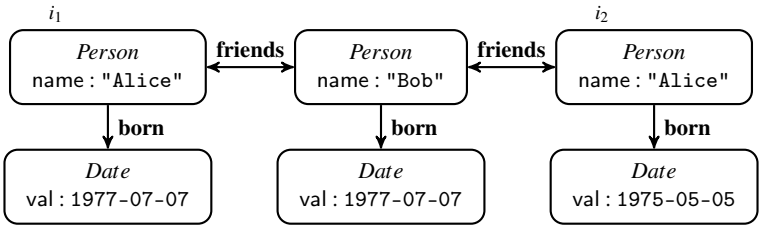


Figure 1: Example of a Data Graph

Proposition 4. $\mathfrak{M},n \Vdash \langle\alpha\rangle\varphi$ *iff exists* $n' \in \mathrm{N}$ *s.t.,* $\mathfrak{M},n,n' \Vdash \alpha$ *and* $\mathfrak{M},n' \Vdash \varphi$. *Moreover,* $\mathfrak{M},n \Vdash [\alpha =_{\mathsf{c}} \beta]$ *iff for all* $n',n'' \in \mathrm{N}$, $\mathfrak{M},n,n' \Vdash \alpha$ *and* $\mathfrak{M},n,n'' \Vdash \beta$ *imply* $n' \approx_{\mathsf{c}} n''$. *Finally,* $\mathfrak{M},n \Vdash [\alpha \neq_{\mathsf{c}} \beta]$ *iff for all* $n',n'' \in \mathrm{N}$, $\mathfrak{M},n,n' \Vdash \alpha$ *and* $\mathfrak{M},n,n'' \Vdash \beta$ *imply* $n' \not\approx_{\mathsf{c}} n''$.

Some intuitive remarks are helpful here. Path expressions indicate reachability relations on nodes, i.e., nodes n and n' satisfy the path expression α if the pair (n,n') is in the relation defined by α; or, equivalently, if n' is reachable from n following an α path. The test expression $\varphi?$ checks whether the node expression φ holds in a given point in a path. The jump-to-key expression i: resets the current path to start from the node whose key is i. Node expressions p, i, $\bot$, $\varphi \to \psi$, $@_i\varphi$, and $\langle \mathsf{a} \rangle \varphi$, have classical readings (see [9]). The novel data comparison operator $\langle \alpha =_{\mathsf{c}} \beta \rangle$ (resp. $\langle \alpha \neq_{\mathsf{c}} \beta \rangle$) requires the existence of paths α and β, from the current point of evaluation, whose end nodes are in the relation $\approx_{\mathsf{c}}$ (resp. $\not\approx_{\mathsf{c}}$) relation. This mirrors how XPath performs data comparisons over data graphs.

We conclude the presentation of $\mathrm{HXPath_D}$ by illustrating how the logic functions as a semi-structured query language over data graphs. The idea is that hybrid data models represent data graphs, while node expressions represent queries that specify both structural properties (e.g., reachability) and data-aware conditions (e.g., comparisons between node values). The following example clarifies this intuition.

Example 1. Fig. 1 shows a simple example of a data graph. We will not formally define data graphs here, see [10] for further details. Intuitively, we can see that the graph encodes information about people and their birth dates, and it also includes a friendship relation. Notice that the graph encodes information of different types: propositional information (certain nodes are defined as Persons, others are Dates), relational information (e.g., the **born** relation encodes the birthdate of a person), and concrete data encoded as pairs *attribute:data* (e.g., one of the persons in the data graph is named `"Alice"`). Moreover, the labels i_1 and i_2 are indexes, that allows direct access to certain nodes in the data graph.

The information in the data graph shown in Fig. 1 can be directly encoded as a hybrid data model. For the structure of the data graph this is immediate. The propositional content in the data graph is captured using proposition symbols, while node indices are managed via the nominal assignment. Crucially, concrete data values are abstracted and represented through data comparisons. Namely, for each attribute c, we introduce in the model a comparison relation $\approx_{\mathsf{c}}$ such that $n_1 \approx_{\mathsf{c}} n_2$ iff there is a data value d such that $\mathsf{c} : \mathsf{d}$ (as shown in the figure) is both in n_1 and n_2—that is, n_1 and n_2 have the same value d in attribute c. The following hybrid data model corresponds to the data graph in Fig. 1.

$$\mathrm{N} = \{n_1,n_2,n_3,n_4,n_5,n_6\} \qquad \mathrm{V}(\mathit{Person}) = \{n_1,n_2,n_3\} \qquad \approx_{\mathsf{name}} = \{(n_1,n_3)\}^{\mathsf{eq}}$$
$$\mathrm{R}_{\textbf{friends}} = \{(n_1,n_2),(n_2,n_1),(n_2,n_3),(n_3,n_2)\} \qquad \mathrm{V}(\mathit{Date}) = \{n_4,n_5,n_6\} \qquad \approx_{\mathsf{val}} = \{(n_4,n_5)\}^{\mathsf{eq}}$$
$$\mathrm{R}_{\textbf{born}} = \{(n_1,n_4),(n_2,n_5),(n_3,n_6)\} \qquad g(i_1) = n_1 \qquad g(i_2) = n_3$$

We use R^{eq} to denote the closure of R under reflexivity, symmetry and transitivity. The equivalence relations $\approx_{\mathsf{name}}$ and $\approx_{\mathsf{val}}$ are sufficient for the type of data operations permitted in $\mathrm{HXPath_D}$.

With the hybrid data model in place, we now turn to how node expressions act as queries. The table below presents a few example properties, along with their formalizations as node expressions:

PROPERTY	FORMALIZATION
The person with key i_1 is a friend of someone born on the same day.	$\langle i_1{:}\,\mathbf{born}\,(\mathit{Date}?) =_{\mathsf{val}} i_1{:}\,\mathbf{friends}\,\mathbf{born}\,(\mathit{Date}?)\rangle$
The person with key i_2 shares a birth date with none of her friends.	$[i_2{:}\,\mathbf{born}\,(\mathit{Date}?) \neq_{\mathsf{val}} i_2{:}\,\mathbf{friends}\,\mathbf{born}\,(\mathit{Date}?)]$
The persons with keys i_1 and i_2 have the same name but different birth dates.	$\langle i_1{:}\,(\mathit{Person}?) =_{\mathsf{name}} i_2{:}\,(\mathit{Person}?)\rangle \wedge \langle i_1{:}\,\mathbf{born}\,(\mathit{Date}?) \neq_{\mathsf{val}} i_2{:}\,\mathbf{born}\,(\mathit{Date}?)\rangle$

Each of the node expressions above evaluates to true at every point in the hybrid data model constructed from the example graph database. This showcases how $\mathrm{HXPath}_{\mathrm{D}}$ combines structural navigation with data-aware comparisons to express rich, semantically meaningful queries.

To sum up, $\mathrm{HXPath}_{\mathrm{D}}$ is of interest both in practice and in theory. On the practical side, it faithfully captures a fragment of XPath involving data-aware queries and key-based navigation. On the theoretical side, it poses new challenges for modal logic: its comparison modalities are complex as they combine navigation modalities with (in)equality reasoning. Despite these challenges, in the next section we show that an elegant sequent calculus for $\mathrm{HXPath}_{\mathrm{D}}$ can indeed be defined.

3 A Sequent Calculus for $\mathrm{HXPath}_{\mathrm{D}}$

In this section, we present our Gentzen-style sequent calculus **G** for $\mathrm{HXPath}_{\mathrm{D}}$. The calculus draws inspiration from the modal system with equality developed in [35], but extends it to accommodate the distinctive features of $\mathrm{HXPath}_{\mathrm{D}}$: nominals, satisfiability modalities, path expressions, and data comparisons. In brief, we build **G** on sequents $\Gamma \vdash \Delta$, where Γ, Δ is a set of node expressions in $\mathrm{HXPath}_{\mathrm{D}}$. Sequents capture the idea that if all node expressions in Γ hold, then at least one node expression in Δ must also hold. The inference rules of **G** systematically decompose the expressions in sequents, closely mirroring the semantic behavior of each logical connective. Out aim is to construct a proof system that addresses the entailment problem for the logic, and to provide a rule-based decomposition of the meaning of its logical connectives. As we will see also, the calculus is particularly well-suited for structural analysis of proofs. With the basic ideas in place, let us begin by: defining sequents, introducing the inference rules, and outlining the construction of derivations in **G**.

Definition 5. *A* sequent *is a pair $\Gamma \vdash \Delta$, where Γ and Δ are finite, possibly empty sets of node expressions of the form $\langle i{:}\,\blacktriangle\, j{:}\rangle$ or $@_i\varphi$.*[2] *In a sequent $\Gamma \vdash \Delta$, the set Γ is called the* antecedent *and the set Δ is called the* consequent*. In turn, a* rule *is a pair $(\Gamma \vdash \Delta, \{\Gamma_1 \vdash \Delta_1, \ldots, \Gamma_n \vdash \Delta_n\})$, where $\Gamma \vdash \Delta$ is called the* conclusion*, and $\{\Gamma_1 \vdash \Delta_1, \ldots, \Gamma_n \vdash \Delta_n\}$ the set of* premisses *of the rule. If the set of premisses is empty, we say that the rule is an* axiom*. In each rule, certain node expressions in the conclusion sequent are designated as* principal*. These are the node expressions the rule acts upon. The remaining node expressions in the sequent are the* context *of the rule, and are carried unchanged across its application. This distinction enables the rule to isolate and analyze the logical structure of the principal formulas while treating the context uniformly. The rules of* **G** *are shown in Fig. 2, in a standard format—i.e., the conclusion is shown under a line, the premisses are above the line in no particular order, the principal node expressions and the context are identified immediately from the presentation.*

[2] Following standard notation, we will not use curly brackets when writing down sequents.

With sequents and rules defined, we proceed to formalize the notion of a derivation in **G**.

Definition 6. *A* derivation *in* **G** *is a sequent-labeled tree constructed according to the rules in Fig. 2. More precisely, each non-leaf node in the tree is a sequent that is the conclusion of a rule in* **G***, its immediate successors in the tree (going upwards) are the premisses of that rule. The root of the tree is called the* end-sequent *of the derivation. A sequent is* derivable *if it is the end-sequent of some derivation, and it is* provable *if it has a derivation whose leaves are all instances of (Ax) or (⊥). We use* $\Gamma \vdash_{\mathbf{G}} \Delta$ *to indicate that* $\Gamma \vdash \Delta$ *is provable. A rule is* derived *iff there is a derivation of the conclusion of the rule whose leaves are either axioms or belong to the premisses of the rule.*

PROPOSITIONAL RULES

$$\frac{}{\varphi,\Gamma\vdash\Delta,\varphi}\,(\mathrm{Ax})^{\ddagger} \qquad \frac{}{@_i\bot,\Gamma\vdash\Delta}\,(\bot) \qquad \frac{\Gamma\vdash\Delta,@_i\varphi \quad @_i\psi,\Gamma\vdash\Delta}{@_i(\varphi\to\psi),\Gamma\vdash\Delta}\,(\to\mathrm{L}) \qquad \frac{@_i\varphi,\Gamma\vdash\Delta,@_i\psi}{\Gamma\vdash\Delta,@_i(\varphi\to\psi)}\,(\to\mathrm{R})$$

‡ φ is of the form $@_i p$, $@_i j$, or $\langle i{:}=_{\mathsf{c}} j{:}\rangle$

RULES FOR NOMINALS

$$\frac{@_i i,\Gamma\vdash\Delta}{\Gamma\vdash\Delta}\,(@\mathrm{T}) \qquad \frac{@_j k,@_i j,@_i k,\Gamma\vdash\Delta}{@_i j,@_i k,\Gamma\vdash\Delta}\,(@5) \qquad \frac{@_i j,\Gamma\vdash\Delta}{\Gamma\vdash\Delta}\,(\mathrm{Nom})^{\dagger}$$

$$\frac{@_j\varphi,@_i j,@_i\varphi,\Gamma\vdash\Delta}{@_i j,@_i\varphi,\Gamma\vdash\Delta}\,(\mathrm{S}_1)^{\ddagger} \qquad \frac{@_i\langle\mathsf{a}\rangle k,@_j k,@_i\langle\mathsf{a}\rangle j,\Gamma\vdash\Delta}{@_j k,@_i\langle\mathsf{a}\rangle j,\Gamma\vdash\Delta}\,(\mathrm{S}_2) \qquad \frac{\langle j{:}=_{\mathsf{c}} k{:}\rangle,@_i j,\langle i{:}=_{\mathsf{c}} k{:}\rangle,\Gamma\vdash\Delta}{@_i j,\langle i{:}=_{\mathsf{c}} k{:}\rangle,\Gamma\vdash\Delta}\,(\mathrm{S}_3)$$

† j is not in the conclusion ‡ φ is of the form p, $\bot$, or $\langle\mathsf{a}\rangle k$

RULES FOR MODALITIES

$$\frac{@_i\varphi,\Gamma\vdash\Delta}{@_j@_i\varphi,\Gamma\vdash\Delta}\,(@\mathrm{L}) \qquad \frac{\Gamma\vdash\Delta,@_i\varphi}{\Gamma\vdash\Delta,@_j@_i\varphi}\,(@\mathrm{R}) \qquad \frac{@_i\langle\mathsf{a}\rangle j,@_j\varphi,\Gamma\vdash\Delta}{@_i\langle\mathsf{a}\rangle\varphi,\Gamma\vdash\Delta}\,(\langle\mathsf{a}\rangle\mathrm{L})^{\dagger} \qquad \frac{@_i\langle\mathsf{a}\rangle j,\Gamma\vdash\Delta,@_i\langle\mathsf{a}\rangle\varphi,@_j\varphi}{@_i\langle\mathsf{a}\rangle j,\Gamma\vdash\Delta,@_i\langle\mathsf{a}\rangle\varphi}\,(\langle\mathsf{a}\rangle\mathrm{R})$$

$$\frac{@_i\langle\alpha\rangle j,@_i\langle\beta\rangle k,\langle j{:}\blacktriangle k{:}\rangle,\Gamma\vdash\Delta}{@_i\langle\alpha\blacktriangle\beta\rangle,\Gamma\vdash\Delta}\,(\langle\blacktriangle\rangle\mathrm{L})^{\ddagger} \qquad \frac{@_i\langle\alpha\rangle j,@_i\langle\beta\rangle k,\Gamma\vdash\Delta,@_i\langle\alpha\blacktriangle\beta\rangle,\langle j{:}\blacktriangle k{:}\rangle}{@_i\langle\alpha\rangle j,@_i\langle\beta\rangle k,\Gamma\vdash\Delta,@_i\langle\alpha\blacktriangle\beta\rangle}\,(\langle\blacktriangle\rangle\mathrm{R})$$

† j is not in the conclusion ‡ j and k are different and not in the conclusion

RULES FOR DATA COMPARISON

$$\frac{\langle i{:}=_{\mathsf{c}} i{:}\rangle,\Gamma\vdash\Delta}{\Gamma\vdash\Delta}\,(\mathrm{EqT}) \qquad \frac{\langle j{:}=_{\mathsf{c}} k{:}\rangle,\langle i{:}=_{\mathsf{c}} j{:}\rangle,\langle i{:}=_{\mathsf{c}} k{:}\rangle,\Gamma\vdash\Delta}{\langle i{:}=_{\mathsf{c}} j{:}\rangle,\langle i{:}=_{\mathsf{c}} k{:}\rangle,\Gamma\vdash\Delta}\,(\mathrm{Eq5}) \qquad \frac{\Gamma\vdash\Delta,\langle i{:}=_{\mathsf{c}} j{:}\rangle}{\langle i{:}\neq_{\mathsf{c}} j{:}\rangle,\Gamma\vdash\Delta}\,(\mathrm{NEqL}) \qquad \frac{\langle i{:}=_{\mathsf{c}} j{:}\rangle,\Gamma\vdash\Delta}{\Gamma\vdash\Delta,\langle i{:}\neq_{\mathsf{c}} j{:}\rangle}\,(\mathrm{NEqR})$$

STRUCTURAL RULES

$$\frac{\Gamma\vdash\Delta,\varphi \quad \varphi,\Gamma'\vdash\Delta'}{\Gamma,\Gamma'\vdash\Delta,\Delta'}\,(\mathrm{Cut}) \qquad \frac{\Gamma\vdash\Delta}{\varphi,\Gamma\vdash\Delta}\,(\mathrm{WL}) \qquad \frac{\Gamma\vdash\Delta}{\Gamma\vdash\Delta,\varphi}\,(\mathrm{WR})$$

Figure 2: Sequent Calculus **G** for $\mathrm{HXPath_D}$.

The inference rules of **G** are organized into five groups, each corresponding to a different fragment of the language of $\mathrm{HXPath_D}$. We briefly explain the rationale behind the rules in each group, highlighting the logical principles and the design choices they reflect.

PROPOSITIONAL RULES. The propositional rules govern implication and falsity, and follow familiar patterns from classical sequent calculi. Note that the node expression φ in the axiom rule (Ax) is restricted in form. This restriction suffices for completeness. The generalized version of the rule where φ is of the form $@_i\psi$ for ψ an arbitrary node expression can be obtained as a derived rule.

RULES FOR NOMINALS. The rules for nominals handle node expressions involving named points in the model, ensuring they behave as references to specific nodes. (@T) and (@5) characterize reflexivity and Euclideanness. The (Nom) rule allows a named node to be given a fresh alias. This plays a technical role in the completeness proof. Finally, the substitution rules (S_m) reflect that when $@_i j$ holds, then i can be substituted by j in different contexts. These rules are inspired by the treatment of equality in the labeled sequent calculus with equality from [35]. Again, we note that all (S_i) are presented in a somewhat restricted form. Specifically, a more general version of (S_1) can be formulated without the restriction on φ. In turn, (S_2) can be generalized to handle arbitrary paths α, not just atomic modalities a. Finally, (S_3) can be generalized to handle arbitrary comparisons $\blacktriangle$, not just data equality. These generalized rules are derivable within the system. Presenting them in their restricted form helps to simplify the system by limiting the scope of interactions between rules, which is particularly beneficial for proving cut elimination.

RULES FOR MODALITIES. The modal rules capture the navigational fragment of the language. (@L) and (@R) reflect the global nature of the satisfiability operator: when the operator is nested, only the inner occurrence is relevant. The diamond rules ($\langle\mathsf{a}\rangle$L) and ($\langle\mathsf{a}\rangle$R) follow standard patterns from modal sequent calculi. In particular, ($\langle\mathsf{a}\rangle$L) introduces fresh nominals as witnesses for the $\langle\mathsf{a}\rangle$ modality. Once again, one can consider generalized versions of the diamond rules in which the atomic modality a is replaced by an arbitrary path expression α. These generalized forms are available as derived rules within the system. Finally, the rules ($\langle\blacktriangle\rangle$L) and ($\langle\blacktriangle\rangle$R) decompose a node expression $@_i\langle\alpha \blacktriangle \beta\rangle$ into its essential components: node expressions $@_i\langle\alpha\rangle j$ and $@_i\langle\beta\rangle k$, capturing the navigation paths, and the atomic comparison $\langle j{:}\ \blacktriangle\ k{:}\rangle$ at the endpoints of the paths. Notice there is a direct analogy between the behavior of these last two rules and the diamond rules.

RULES FOR DATA COMPARISON. Data comparison rules handle equality and inequality between data values at the endpoints of paths. (EqT) and (Eq5) express that data equality is an equivalence relation, while (NeqL) and (NeqR) allow inequality to be reasoned about in terms of equality.

STRUCTURAL RULES. Finally, the structural rules govern manipulation of the sequent structure itself. These rules play a central role in the meta-theoretical analysis of the system. We will show in Sec. 4, however, that these rules can be eliminated.

3.1 Soundness

We now turn to the proof of soundness for **G**. We define a formal notion of sequent validity in the context of hybrid data models and then verify, via a case-by-case analysis, that each inference rule of **G** preserves this notion of validity. This strategy ensures that all provable sequents are valid. We begin by formally defining the semantics of sequents in terms of the satisfaction relation introduced earlier.

Definition 7. *A sequent $\Gamma \vdash \Delta$ is* valid *iff for all $\mathfrak{M}$, it follows that $\mathfrak{M} \Vdash \Gamma$ implies $\mathfrak{M} \Vdash \psi$ for some $\psi \in \Delta$. A rule preserves validity iff the validity of the premisses of the rule implies the validity of the conclusion of the rule.*

Lemma 8 (Soundness)**.** *Every rule in* **G** *preserves validity.*

Proof. We present a selection of representative cases below. The remaining cases use a similar argument and can be verified by routine inspection. In all cases below we reason by contradiction.

(Nom) Let $\mathfrak{A}$ be a model s.t. $\mathfrak{A} \Vdash \Gamma$ and $\mathfrak{A} \nVdash \psi$ for all $\psi \in \Delta$. Then, introduce a new nominal j that is not in Γ, Δ and build a model $\mathfrak{B}$ that is just like $\mathfrak{A}$ with the exception that $\mathfrak{B} \Vdash @_i j$. We have $\mathfrak{B} \Vdash @_i j, @_i\varphi, \Gamma$ and $\mathfrak{B} \nVdash \psi$ for all $\psi \in \Delta$. This contradicts the validity of the premiss of the rule.

($\langle\blacktriangle\rangle$L) We have: (1) $\blacktriangle$ is $=_{\mathsf{c}}$, or (2) $\blacktriangle$ is $\neq_{\mathsf{c}}$. For (1), take any model $\mathfrak{A}$ s.t.: $\mathfrak{A} \Vdash @_i\langle\alpha =_{\mathsf{c}} \beta\rangle, \Gamma$, and $\mathfrak{A} \nVdash \psi$ for all $\psi \in \Delta$. The semantics of $@_i\langle\alpha =_{\mathsf{c}} \beta\rangle$ tells us there are n and n' in $\mathfrak{A}$ s.t.: $\mathfrak{A}, g(i), n \Vdash \alpha, \mathfrak{A}, g(i), n' \Vdash \beta$, and $(n, n') \in \approx_{\mathsf{c}}$. Then, choose nominals j and k that do not appear in $\Gamma, \Delta, \alpha, \beta$ and build a model $\mathfrak{B}$ that is identical to $\mathfrak{A}$ with the exception that $\mathfrak{B}, n \Vdash j$ and $\mathfrak{B}, n' \Vdash k$. It is clear that $\mathfrak{B} \Vdash @_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\, =_{\mathsf{c}} k{:}\rangle, \Gamma$ and $\mathfrak{B} \nVdash \psi$ for all $\psi \in \Delta$. This contradicts the validity of the premiss of the rule. The case for (2) is similar.

($\langle\blacktriangle\rangle$R) We have: (1) $\blacktriangle$ is $=_{\mathsf{c}}$, or (2) $\blacktriangle$ is $\neq_{\mathsf{c}}$. For (1), take any model $\mathfrak{A}$ s.t.: $\mathfrak{A} \Vdash @_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \Gamma$ and $\mathfrak{B} \nVdash \psi$ for all $\psi \in \Delta, @_i\langle\alpha =_{\blacktriangle} \beta\rangle$. In particular, $\mathfrak{A} \nVdash @_i\langle\alpha =_{\mathsf{c}} \beta\rangle$. This means that for all n and n' in $\mathfrak{A}$, it follows that $\mathfrak{A}, g(i), n \Vdash \alpha$, $\mathfrak{A}, g(i), n' \Vdash \beta$, and $(n, n') \notin \approx_{\mathsf{c}}$. This contradicts the validity of premiss of the rule. The case for (2) is similar. □

Soundness of **G** follows from Lemma 8 by induction on the structure of a derivation of a sequent.

Theorem 9 (Soundness)**.** *Every provable sequent in* **G** *is valid, i.e.,* $\Gamma \vdash_{\mathbf{G}} \Delta$ *implies* $\Gamma \vDash \Delta$.

3.2 Invertibility of Rules

Having established the soundness of the calculus **G**, we prove one of its key properties: all inference rules are invertible. Invertibility plays a central role in the proof-theoretic analysis of a sequent calculus. It allows backward reasoning—from a conclusion to its premisses—without loss of validity. Invertible rules are also particularly well-suited for proof search procedures (see, e.g., [35]). In our case, invertibility also plays a role in our proof of completeness of **G**, where we rely on the ability to apply certain rules in reverse to construct derivations..

Definition 10. *A rule* (P) *is* invertible *iff there is a derivation of each premiss of the rule whose leaves are either axioms or the conclusion of the rule. Any such derivation is called an* inverse *of the rule and is denoted by* (P^{-1}).

Theorem 11. *Every rule in* **G** *is invertible.*

Proof. Invertibility of propositional rules is standard. For (@T), (@5), (Nom), (S_1), (S_2), (S_3), ($\langle\mathsf{a}\rangle$R), ($\langle\blacktriangle\rangle$R), (EqT), and (Eq5) invertibility follows by weakening. We show (@L^{-1}), ($\langle\mathsf{a}\rangle\mathrm{L}^{-1}$), and ($\langle\blacktriangle\rangle\mathrm{L}^{-1}$).[3]
Case (@L^{-1}): Given a derivation of $@_i\varphi, \Gamma \vdash \Delta$, we build a derivation of $@_i\varphi$ as:

$$\dfrac{\dfrac{\dfrac{}{@_i\varphi, \Gamma \vdash \Delta, @_i\varphi}\,(\mathrm{Ax})}{@_i\varphi, \Gamma \vdash \Delta, @_j@_i\varphi}\,(@\mathrm{R}) \qquad @_j@_i\varphi, \Gamma \vdash \Delta}{@_i\varphi, \Gamma \vdash \Delta}\,(\mathrm{Cut})$$

Case ($\langle\mathsf{a}\rangle\mathrm{L}^{-1}$): Given a derivation of $@_i\langle\mathsf{a}\rangle\varphi, \Gamma \vdash \Delta$, we build a derivation of $@_i\langle\mathsf{a}\rangle j, @_j\varphi, \Gamma \vdash \Delta$ as:

$$\dfrac{\dfrac{\dfrac{}{@_i\langle\mathsf{a}\rangle j, @_j\varphi, \Gamma \vdash \Delta, @_i\langle\mathsf{a}\rangle\varphi, @_j\varphi}\,(\mathrm{Ax})}{@_i\langle\mathsf{a}\rangle j, @_j\varphi, \Gamma \vdash \Delta, @_i\langle\mathsf{a}\rangle\varphi}\,(\langle\mathsf{a}\rangle\mathrm{R}) \qquad @_i\langle\mathsf{a}\rangle\varphi, \Gamma \vdash \Delta}{@_i\langle\mathsf{a}\rangle j, @_j\varphi, \Gamma \vdash \Delta}\,(\mathrm{Cut})$$

Case ($\langle\blacktriangle\rangle$L): Given a derivation of $@_i\langle\alpha \blacktriangle \beta\rangle, \Gamma \vdash \Delta$, we derive $@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\, \blacktriangle k{:}\rangle, \Gamma \vdash \Delta$ as:

$$\dfrac{\dfrac{\dfrac{}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\, \blacktriangle k{:}\rangle, \Gamma \vdash \Delta, @_i\langle\alpha \blacktriangle \beta\rangle, \langle j{:}\, \blacktriangle k{:}\rangle}\,(\mathrm{Ax})}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\, \blacktriangle k{:}\rangle, \Gamma \vdash \Delta, @_i\langle\alpha \blacktriangle \beta\rangle}\,(\langle\blacktriangle\rangle\mathrm{R}) \qquad @_i\langle\alpha \blacktriangle \beta\rangle, \Gamma \vdash \Delta}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\, \blacktriangle k{:}\rangle, \Gamma \vdash \Delta}\,(\mathrm{Cut})$$

□

[3] We use colors to guide derivations: cyan for cut expressions, pink for principal, and magenta for other relevant elements.

3.3 Completeness

In this section, we establish the completeness of **G**. Rather than constructing canonical models, we prove completeness by way of a Hilbert-style axiomatization for $\mathrm{HXPath_D}$, which we refer to as **H**. The axiomatization, introduced in [10], has been shown to be both sound and complete. Concretely, we demonstrate that every theorem of **H** corresponds to a provable sequent in **G**. This ensures that all semantically valid sequents are derivable in the sequent calculus. Our strategy underscores the expressiveness of **G**, and leverages existing results. As a further benefit, we reveal a close correspondence between axiomatic and sequent-based reasoning in $\mathrm{HXPath_D}$.

Lemma 12. *Let $\vdash^n_{\mathbf{H}} \varphi$ indicate that φ is a theorem in* **H***, with a derivation of length n. In addition, let i be a nominal not in φ. It follows that, for any n, $\vdash^n_{\mathbf{H}} \varphi$ implies $\vdash_{\mathbf{G}} @_i\varphi$ (i.e, $\vdash @_i\varphi$ is provable in* **G***).*

Proof. The proof proceeds by (strong) induction on the length of a derivation in **H**. The base case requires that each axiom φ in **H** has a corresponding provable sequent $\vdash_{\mathbf{G}} @_i\varphi$. To simplify these proofs we will make use of the following derived rules in **G**.

Derived Rules. We introduce derived rules for treating other propositional connectives as if they were primitive. Notably,

$$\frac{@_i\top, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\,(\top\mathrm{L}) \qquad \frac{@_i\varphi, @_i\psi, \Gamma \vdash \Delta}{@_i(\varphi \wedge \psi), \Gamma \vdash \Delta}\,(\wedge\mathrm{L}) \qquad \frac{\Gamma \vdash \Delta, @_i\psi \quad \Gamma \vdash \Delta, @_i\varphi}{\Gamma \vdash \Delta, @_i(\varphi \wedge \psi)}\,(\wedge\mathrm{R}) \qquad \frac{@_i\varphi, \Gamma \vdash \Delta, @_i\psi \quad @_i\psi, \Gamma \vdash \Delta, @_i\varphi}{\Gamma \vdash \Delta, @_i(\varphi \leftrightarrow \psi)}\,(\leftrightarrow\mathrm{R})$$

As mentioned earlier, we introduce a generalized form of (Ax) as a derived rule.

$$\frac{}{@_i\varphi, \Gamma \vdash \Delta, @_i\varphi}\,(\mathrm{AxG})$$

In (AxG) φ is an arbitrary node expression. When no confusion arises, we will also refer to this rule simply as (Ax). Finally, we introduce derived rules characterizing the symmetry of ▲ explicitly. Precisely,

$$\frac{\langle j{:}\blacktriangle\, i{:}\rangle, \Gamma \vdash \Delta}{\langle i{:}\blacktriangle\, j{:}\rangle, \Gamma \vdash \Delta}\,(\langle\blacktriangle\rangle\mathrm{B}).$$

Base Case. We are now ready to present derivations for selected axioms of **H**. Beyond their technical role, these derivations also serve an explanatory purpose: they illustrate how the rules of **G** reveal the semantic behavior of logical connectives, offering insight into their proof-theoretic interpretation and clarifying the structure of the corresponding axioms. We focus on the axioms (equal), (▲-comm), and (ε-trans) expressing the equivalence properties of data equality: reflexivity, symmetry, and transitivity. These axioms are listed below for reference.

$$\begin{aligned}
&\text{(equal)}\ \langle \varepsilon =_{\mathsf{c}} \varepsilon \rangle \\
&\text{(▲-comm)}\ \langle \alpha \blacktriangle \beta \rangle \leftrightarrow \langle \beta \blacktriangle \alpha \rangle \\
&\text{(ε-trans)}\ \langle \alpha =_{\mathsf{c}} \varepsilon \rangle \wedge \langle \varepsilon =_{\mathsf{c}} \beta \rangle \rightarrow \langle \alpha =_{\mathsf{c}} \beta \rangle.
\end{aligned}$$

For (equal), we must show $\vdash_{\mathbf{G}} @_i\langle \varepsilon =_{\mathsf{c}} \varepsilon \rangle$. For (▲-comm), we must show $\vdash_{\mathbf{G}} @_i(\langle \alpha \blacktriangle \beta \rangle \leftrightarrow \langle \beta \blacktriangle \alpha \rangle)$. Finally, for ($\varepsilon$-trans), we must show $\vdash_{\mathbf{G}} @_i(\langle \alpha =_{\mathsf{c}} \varepsilon \rangle \wedge \langle \varepsilon =_{\mathsf{c}} \beta \rangle \rightarrow \langle \alpha =_{\mathsf{c}} \beta \rangle)$. We deal with each of these cases individually.

Reflexivity. The derivation below establishes $\vdash_{\mathbf{G}} @_i\langle \varepsilon =_{\mathsf{c}} \varepsilon \rangle$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\langle i{:}=_{\mathsf{c}} i{:}\rangle, @_i\langle\varepsilon\rangle i \vdash @_i\langle\varepsilon=_{\mathsf{c}}\varepsilon\rangle, \langle i{:}=_{\mathsf{c}} i{:}\rangle}\,(\text{Ax})}{@_i\langle\varepsilon\rangle i \vdash @_i\langle\varepsilon=_{\mathsf{c}}\varepsilon\rangle, \langle i{:}=_{\mathsf{c}} i{:}\rangle}\,(\text{EqT})}{@_i\langle\varepsilon\rangle i \vdash @_i\langle\varepsilon=_{\mathsf{c}}\varepsilon\rangle}\,(\langle\blacktriangle\rangle\text{R})}{@_i\top, @_i i \vdash @_i\langle\varepsilon=_{\mathsf{c}}\varepsilon\rangle}\,(\wedge\text{L}^{-1})}{@_i i \vdash @_i\langle\varepsilon=_{\mathsf{c}}\varepsilon\rangle}\,(\top\text{L})}{\vdash @_i\langle\varepsilon=_{\mathsf{c}}\varepsilon\rangle}\,(@\text{T})$$

When reading the derivation bottom-up, most of the effort goes into constructing an empty path on the antecedent side of the sequent. Intuitively, the empty path encodes the idea that we remain at the current node. We then use ($\langle\blacktriangle\rangle$R) to compare the data at the node and itself. The derivation concludes with the rule (EqT) just before (Ax), which captures the reflexivity of data equality in **G**.

Symmetry. The next derivation establishes $\vdash_{\mathbf{G}} @_i(\langle\alpha\blacktriangle\beta\rangle\leftrightarrow\langle\beta\blacktriangle\alpha\rangle)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle k{:}\blacktriangle j{:}\rangle \vdash @_i\langle\beta\blacktriangle\alpha\rangle, \langle k{:}\blacktriangle j{:}\rangle}\,(\text{Ax})}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\blacktriangle k{:}\rangle \vdash @_i\langle\beta\blacktriangle\alpha\rangle, \langle k{:}\blacktriangle j{:}\rangle}\,(\langle\blacktriangle\rangle\text{B})}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\blacktriangle k{:}\rangle \vdash @_i\langle\beta\blacktriangle\alpha\rangle}\,(\langle\blacktriangle\rangle\text{R})}{@_i\langle\alpha\blacktriangle\beta\rangle \vdash @_i\langle\beta\blacktriangle\alpha\rangle}\,(\langle\blacktriangle\rangle\text{L})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{@_i\langle\beta\rangle k, @_i\langle\alpha\rangle j, \langle j{:}\blacktriangle k{:}\rangle \vdash @_i\langle\alpha\blacktriangle\beta\rangle, \langle j{:}\blacktriangle k{:}\rangle}\,(\text{Ax})}{@_i\langle\beta\rangle k, @_i\langle\alpha\rangle j, \langle k{:}\blacktriangle j{:}\rangle \vdash @_i\langle\alpha\blacktriangle\beta\rangle, \langle j{:}\blacktriangle k{:}\rangle}\,(\langle\blacktriangle\rangle\text{B})}{@_i\langle\beta\rangle k, @_i\langle\alpha\rangle j, \langle k{:}\blacktriangle j{:}\rangle \vdash @_i\langle\alpha\blacktriangle\beta\rangle}\,(\langle\blacktriangle\rangle\text{R})}{@_i\langle\beta\blacktriangle\alpha\rangle \vdash @_i\langle\alpha\blacktriangle\beta\rangle}\,(\langle\blacktriangle\rangle\text{L})$$

$$\dfrac{@_i\langle\alpha\blacktriangle\beta\rangle \vdash @_i\langle\beta\blacktriangle\alpha\rangle \qquad @_i\langle\beta\blacktriangle\alpha\rangle \vdash @_i\langle\alpha\blacktriangle\beta\rangle}{\vdash @_i(\langle\alpha\blacktriangle\beta\rangle\leftrightarrow\langle\beta\blacktriangle\alpha\rangle)}\,(\leftrightarrow\text{R})$$

As before, reading the derivation bottom-up, we observe that we first construct the appropriate paths in the antecedent. These paths serve to navigate to the nodes whose data values we aim to compare. As mentioned, in **G** the derived rule ($\langle\blacktriangle\rangle$B) captures symmetry. The derivation concludes with an application of this rule before (Ax).

Transitivity. The final derivation establishes $\vdash_{\mathbf{G}} @_i(\langle\alpha=_{\mathsf{c}}\varepsilon\rangle\wedge\langle\varepsilon=_{\mathsf{c}}\beta\rangle\rightarrow\langle\alpha=_{\mathsf{c}}\beta\rangle)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\langle a{:}=_{\mathsf{c}} b{:}\rangle, \langle c{:}=_{\mathsf{c}} a{:}\rangle, \langle c{:}=_{\mathsf{c}} b{:}\rangle \vdash \langle a{:}=_{\mathsf{c}} b{:}\rangle}\,(\text{Ax})}{\langle c{:}=_{\mathsf{c}} a{:}\rangle, \langle c{:}=_{\mathsf{c}} b{:}\rangle \vdash \langle a{:}=_{\mathsf{c}} b{:}\rangle}\,(\text{Eq5})}{\langle a{:}=_{\mathsf{c}} c{:}\rangle, \langle c{:}=_{\mathsf{c}} b{:}\rangle \vdash \langle a{:}=_{\mathsf{c}} b{:}\rangle}\,(\langle\blacktriangle\rangle\text{B})}{@_d c, \langle a{:}=_{\mathsf{c}} c{:}\rangle, \langle d{:}=_{\mathsf{c}} b{:}\rangle \vdash \langle a{:}=_{\mathsf{c}} b{:}\rangle}\,(\text{S}_3,\text{WL})}{@_i c, @_i d, @_i\top, \langle a{:}=_{\mathsf{c}} c{:}\rangle, @_i\langle\alpha\rangle a, @_i\langle\beta\rangle b, \langle d{:}=_{\mathsf{c}} b{:}\rangle \vdash @_i\langle\alpha=_{\mathsf{c}}\beta\rangle, \langle a{:}=_{\mathsf{c}} b{:}\rangle}\,(@5,\text{WL})$$

$$\dfrac{@_i\langle\alpha\rangle a, @_i\langle\varepsilon\rangle c, \langle a{:}=_{\mathsf{c}} c{:}\rangle, @_i\langle\varepsilon\rangle d, @_i\langle\beta\rangle b, \langle d{:}=_{\mathsf{c}} b{:}\rangle \vdash @_i\langle\alpha=_{\mathsf{c}}\beta\rangle}{@_i\langle\alpha=_{\mathsf{c}}\varepsilon\rangle, @_i\langle\varepsilon=_{\mathsf{c}}\beta\rangle \vdash @_i\langle\alpha=_{\mathsf{c}}\beta\rangle}\,(\wedge\text{L},\langle\blacktriangle\rangle\text{R})\,(\langle\blacktriangle\rangle\text{L})$$

$$\dfrac{}{\vdash @_i(\langle\alpha=_{\mathsf{c}}\varepsilon\rangle\wedge\langle\varepsilon=_{\mathsf{c}}\beta\rangle\rightarrow\langle\alpha=_{\mathsf{c}}\beta\rangle)}\,(\rightarrow\text{R},\wedge\text{L})$$

The key idea is to chain two data equalities together. The derivation (going again bottom-up) begins by using empty paths ε to refer to the intermediate node that links α and β. Because of the side conditions on the rules being used, this intermediate node ends up having two names c and d. The rule (@5) indicates c and d are alias. Given this information, the rule (S_3) replaces d by c in $\langle d{:}=_{\mathsf{c}} b{:}\rangle$, yielding $\langle c{:}=_{\mathsf{c}} b{:}\rangle$. The rule ($\langle\blacktriangle\rangle$B) prepares the sequent for the application of (Eq5), which captures transitivity in **G**, before ending with (Ax).

To sum up, (equal), ($\blacktriangle$-comm), and (ε-trans) are introduced in [10] to express the equivalence properties of data equality. Arguably, rules (EqT) and (Eq5) make these properties more transparent. The derivation in **G** of all axioms in **H** can be found in [11].

Inductive Case. We must show that $\vdash^m_{\mathbf{H}}\psi$ implies $\vdash_{\mathbf{G}} @_i\psi$. The inductive hypothesis (IH) is: for all $1\leq n<m$, if $\vdash^n_{\mathbf{H}}\varphi$, then $\vdash_{\mathbf{G}} @_i\varphi$, for some nominal i not occurring in φ. We proceed by cases, depending

on whether $\vdash^m_{\mathbf{H}} \psi$ is obtained using one of the four inference rules in **H**: (MP), (Nec), (Name), or (Paste). The first three cases follow directly; details can be found in [11]. Let us discuss the (Paste) rule:

$$\frac{\vdash @_j\langle \mathsf{a}\rangle k \wedge \langle k{:}\alpha \blacktriangle \beta\rangle \to \chi}{\vdash \langle j{:}\mathsf{a}\alpha \blacktriangle \beta\rangle \to \chi} \text{ where } j \text{ and } k \text{ are different and not in } \chi, \alpha, \beta.$$

An application of this rule assumes that for $n < m$ $\vdash^n_{\mathbf{H}} (@_j\langle \mathsf{a}\rangle k \wedge \langle k{:}\alpha \blacktriangle \beta\rangle) \to \chi$. From the IH, $\vdash_{\mathbf{G}} @_i((@_j\langle \mathsf{a}\rangle k \wedge \langle k{:}\alpha \blacktriangle \beta\rangle) \to \chi)$. The next derivation establishes $\vdash_{\mathbf{G}} @_i(\langle j{:}\mathsf{a}\alpha \blacktriangle \beta\rangle \to \chi)$, as required.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{@_j\langle \mathsf{a}\rangle k \vdash @_j\langle \mathsf{a}\rangle k}(\mathrm{Ax})}{@_j\langle \mathsf{a}\rangle k \vdash @_i@_j\langle \mathsf{a}\rangle k}(@\mathrm{R}) \quad \dfrac{\dfrac{}{\langle a{:}\blacktriangle b{:}\rangle \vdash \langle a{:}\blacktriangle b{:}\rangle}(\mathrm{Ax})}{@_i\langle k{:}\alpha\rangle a, @_i\langle\beta\rangle b, \langle a{:}\blacktriangle b{:}\rangle \vdash @_i\langle k{:}\alpha \blacktriangle \beta\rangle}(\langle\blacktriangle\rangle\mathrm{R,W}*)}{@_k\langle\alpha\rangle a, @_i\langle\beta\rangle b, \langle a{:}\blacktriangle b{:}\rangle \vdash @_i\langle k{:}\alpha \blacktriangle \beta\rangle}(@\mathrm{L}^{-1})}{@_j\langle \mathsf{a}\rangle k, @_k\langle\alpha\rangle a, @_i\langle\beta\rangle b, \langle a{:}\blacktriangle b{:}\rangle \vdash @_i(@_j\langle \mathsf{a}\rangle k \wedge \langle k{:}\alpha \blacktriangle \beta\rangle)}(\wedge\mathrm{R,W}*) \quad \dfrac{\vdash_{\mathbf{G}} @_i(@_j\langle \mathsf{a}\rangle k \wedge \langle k{:}\alpha \blacktriangle \beta\rangle \to \chi)}{@_i(@_j\langle \mathsf{a}\rangle k \wedge \langle k{:}\alpha \blacktriangle \beta\rangle) \vdash @_i\chi}(\to\mathrm{R}^{-1})}{@_j\langle \mathsf{a}\rangle k, @_k\langle\alpha\rangle a, @_i\langle\beta\rangle b, \langle a{:}\blacktriangle b{:}\rangle \vdash @_i\chi}(\mathrm{Cut})$$
$$\dfrac{@_j\langle \mathsf{a}\rangle k, @_k\langle\alpha\rangle a, @_i\langle\beta\rangle b, \langle a{:}\blacktriangle b{:}\rangle \vdash @_i\chi}{@_j\langle \mathsf{a}\alpha\rangle a, @_i\langle\beta\rangle b, \langle a{:}\blacktriangle b{:}\rangle \vdash @_i\chi}(\langle\mathsf{a}\rangle\mathrm{L})$$
$$\dfrac{@_j\langle \mathsf{a}\alpha\rangle a, @_i\langle\beta\rangle b, \langle a{:}\blacktriangle b{:}\rangle \vdash @_i\chi}{@_i\langle j{:}\mathsf{a}\alpha\rangle a, @_i\langle\beta\rangle b, \langle a{:}\blacktriangle b{:}\rangle \vdash @_i\chi}(@\mathrm{L})$$
$$\dfrac{@_i\langle j{:}\mathsf{a}\alpha\rangle a, @_i\langle\beta\rangle b, \langle a{:}\blacktriangle b{:}\rangle \vdash @_i\chi}{@_i\langle j{:}\mathsf{a}\alpha \blacktriangle \beta\rangle \vdash @_i\chi}(\langle\blacktriangle\rangle\mathrm{L})$$
$$\dfrac{@_i\langle j{:}\mathsf{a}\alpha \blacktriangle \beta\rangle \vdash @_i\chi}{\vdash_{\mathbf{G}} @_i(\langle j{:}\mathsf{a}\alpha \blacktriangle \beta\rangle \to \chi)}(\to\mathrm{R})$$

In the derivation above, (W$*$) indicates the simultaneous application of (WL) and (WR). □

Theorem 13 (Completeness)**.** *Every valid sequent is provable.*

Proof. Suppose $\gamma_1, \ldots, \gamma_n \vdash \delta_1, \ldots, \delta_m$ is valid. From the completeness result in [10], we know there is k such that $\vdash^k_{\mathbf{H}} \bigwedge_{1\le i\le n}\gamma_i \to \bigvee_{1\le j\le m}\delta_j$. From Lemma 12, we get $\vdash_{\mathbf{G}} @_i(\bigwedge_{1\le i\le n}\gamma_i \to \bigvee_{1\le j\le m}\delta_j)$. This implies $@_i\gamma_1, \ldots, @_i\gamma_n \vdash_{\mathbf{G}} @_i\delta_1, \ldots, @_i\delta_m$, and so $\gamma_1, \ldots, \gamma_n \vdash_{\mathbf{G}} \delta_1, \ldots, \delta_m$ as required. □

4 Cut Elimination

The rule (Cut) plays a crucial role in our proof of the completeness of **G**. Specifically, it allows us to compose translations of Hilbert-style derivations within **G**. Despite its utility, however, (Cut) can be eliminated. The general strategy, following [35], is to push applications of (Cut) "upwards" in the derivation tree until they disappear at the level of leaves. To this end, we perform an induction on the *size* of the *active cut expression*, with a sub-induction on the *(Cut) height*. We introduce these notions and related results below.

Definition 14. *The* size *of a node expression is defined by mutual recursion as:*

$$\begin{array}{lll}
\mathrm{size}(p) = 1 & \mathrm{size}(\varphi \to \psi) = 1 + \mathrm{size}(\varphi) + \mathrm{size}(\psi) & \mathrm{size}(\mathsf{a}) = 1 \\
\mathrm{size}(i) = 1 & \mathrm{size}(@_i\varphi) = 1 + \mathrm{size}(\varphi) & \mathrm{size}(i{:}) = 1 \\
\mathrm{size}(\bot) = 1 & \mathrm{size}(\langle \mathsf{a}\rangle\varphi) = 1 + \mathrm{size}(\varphi) & \mathrm{size}(\varphi?) = 1 + \mathrm{size}(\varphi) \\
 & \mathrm{size}(\langle \alpha \blacktriangle \beta\rangle) = 1 + \mathrm{size}(\alpha) + \mathrm{size}(\beta) & \mathrm{size}(\alpha\beta) = \mathrm{size}(\alpha) + \mathrm{size}(\beta)
\end{array}$$

The function size *induces a well-founded partial order over the set of node expressions.*

Definition 15. *The* height *of a derivation is the length of its longest branch (e.g., a derivation consisting of only an application of (Ax) has height 1). If $\Gamma \vdash \Delta$ is the end-sequent in a derivation, we will use $\Gamma \vdash^n \Delta$ to indicate that the derivation has height n. The* cut height *of an application of (Cut) in a given derivation is the sum of the heights of the derivations of the premisses of the rule; i.e., if we have derivations $\Gamma \vdash^n \Delta, \varphi$, and $\varphi, \Gamma' \vdash^m \Delta'$, using (Cut), we obtain a derivation $\Gamma, \Gamma' \vdash^{(\max(n,m)+1)} \Delta, \Delta'$. In this case, the cut height is $n+m$. In such an application of (Cut), we call φ the* active cut *expression.*

Intuitively, the *cut height* measures how close to the leaves of a derivation a particular application of (Cut) occurs, taking into consideration the derivations of *both* premisses of the rule. This will be important in our proof of cut elimination. We can now state and prove the main result of this section.

Theorem 16. *Every use of (Cut) in the derivation of a provable sequent can be eliminated.*

Proof. The proof is by induction on two measures: the size of the active cut expression, and the cut height. More precisely, in the proof we associate with each application of (Cut) in a derivation a pair (k,h), called *cut complexity*, where: k corresponds to the size of the active cut expression, and h corresponds to the cut height. The induction is on the lexicographic order of the pairs (k,h).

Base Case. The base cases of the induction are relatively direct. They involve derivations in which (Cut) is applied only once, with axiom rules applied to its premisses, i.e., the premisses of (Cut) must be instances of (Ax) or ($\bot$). Eliminating (Cut) in such configurations is unproblematic. We illustrate one representative case below. Suppose $\mathscr{C}$ and $\mathscr{A}$ are derivations with the following structure:

$$\mathscr{C} = \dfrac{\dfrac{}{\Gamma \vdash \Delta, \varphi}\,(\text{Ax}) \quad \dfrac{}{\varphi, \Gamma' \vdash \Delta'}\,(\bot)}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\text{Cut}) \qquad\qquad \mathscr{A} = \dfrac{}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\text{P})$$

The derivation $\mathscr{C}$ represents a case in which one of the premisses of (Cut) is (Ax) and the other is ($\bot$). This derivation can be transformed into the cut-free derivation $\mathscr{A}$ in which: (P) is (Ax) if $\varphi \notin \Gamma$; and it is ($\bot$) otherwise. The remaining cases use a similar argument.

Inductive Step. The key idea is to identify an application of (Cut) that is minimal according to the cut height, and eliminate it. More precisely, we proceed by considering in a derivation a sub-derivation ending in an application of (Cut) with complexity (k,h) where the value for h is minimal. This guarantees that no other instance of (Cut) appears above it in the sub-derivation. We show that this sub-derivation can be replaced by an alternative one which uses only (Cut) instances whose associated pairs (k',h') are strictly smaller than (k,h). We can then invoke a (strong) inductive hypothesis to claim that these (Cut) instances can also be eliminated.

We proceed by a case analysis based on the syntactic form of the active cut, and on whether the active cut is principal in either premiss of the application of (Cut). This analysis guides a systematic transformation of the derivation where: we push the (Cut) upwards, or replace it with applications of (Cut) involving smaller active cut expressions. In either case, the process eventually leads to a cut-free derivation. We illustrate how this transformation unfolds in some representative scenarios.

Non-principal Cases. First, let us cover the case where the active cut is not principal in the right premiss of (Cut). In this case, the application of (Cut) is permuted up. To illustrate this process, consider, e.g., the derivation:

$$\dfrac{\Gamma \vdash^n \Delta, \varphi \qquad \dfrac{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\blacktriangle k{:}\rangle, \varphi, \Gamma' \vdash^m \Delta}{\varphi, @_i\langle\alpha \blacktriangle \beta\rangle, \Gamma' \vdash \Delta}\,(\langle\blacktriangle\rangle\text{L})}{@_i\langle\alpha \blacktriangle \beta\rangle, \Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\text{Cut})$$

Suppose that in this derivation, the use of (Cut) has an associated complexity $(\text{size}(\varphi), n+(m+1))$, where $n+(m+1)$ is minimal. W.l.o.g., assume that j and k do not appear in $\Gamma \vdash \Delta$.[4] We transform this

[4] If j or k do appear in $\Gamma \vdash \Delta$, we can simply choose different nominals, and rewrite the derivation of $\varphi, @_i\langle\alpha \blacktriangle \beta\rangle, \Gamma' \vdash \Delta$ using the new selection of nominals.

derivation into:

$$\dfrac{\dfrac{\Gamma \vdash^n \Delta, \varphi \qquad \varphi, @_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\blacktriangle k{:}\rangle, \Gamma' \vdash^m \Delta}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\blacktriangle k{:}\rangle, \Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\text{Cut})}{@_i\langle\alpha \blacktriangle \beta\rangle, \Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\langle\blacktriangle\rangle\text{L})$$

The application of (Cut) in the transformed derivation has complexity $(\text{size}(\varphi), n+m)$, so it can be eliminated by the inductive hypothesis.

The remaining cases where the active cut is not principal in the right premiss follow the same strategy. To see why, note that all such derivations have the following general structure:

$$\dfrac{\dfrac{\Gamma \vdash^n \Delta, \varphi \qquad \varphi, \Phi, \Gamma' \vdash \Delta', \Sigma}{\Phi, \Gamma, \Gamma' \vdash \Delta, \Delta', \Sigma}\,(\text{Cut})}{\varphi, \Phi', \Gamma' \vdash^m \Delta', \Sigma'}\,(\text{P})$$

In this derivation, we are considering (P) is a single premiss rule of **G**, the set $\varphi, \Gamma', \Delta'$ is the context for the rule, and the set $\Phi', \Phi, \Sigma', \Sigma$ are the node expressions the rule acts upon. Again, we assume that (Cut) has complexity $(\text{size}(\varphi), n+(m+1))$ where $n+(m+1)$ is minimal; i.e., where (Cut) is not used in $\Gamma \vdash^n \Delta, \varphi$, nor in $\varphi, \Phi', \Gamma' \vdash^m \Delta', \Sigma'$. Modulo a possible renaming of nominals, we transform this derivation into:

$$\dfrac{\dfrac{\Gamma \vdash^n \Delta, \varphi \quad \varphi, \Phi', \Gamma' \vdash^m \Delta', \Sigma'}{\Phi', \Gamma', \Gamma \vdash \Delta, \Delta', \Sigma'}\,(\text{Cut})}{\Phi, \Gamma', \Gamma \vdash \Delta, \Delta', \Sigma}\,(\text{P})$$

The use of (Cut) in the transformed derivation has complexity $(\text{size}(\varphi), n+m)$, using the inductive hypothesis, we obtain a cut-free derivation of $\Phi, \Gamma', \Gamma \vdash \Delta, \Delta', \Sigma$.

The cases where the active cut is not principal in the left premiss is symmetric. The $(\to\text{L})$ case—the only two-premise rule in **G**—is handled similarly, and is well known in the literature.

Principal Cases. Let us now turn our attention to derivations where the active cut is principal in both premisses. Such cases are central to the proof and require careful handling to ensure that the application of (Cut) can still be pushed upwards, and ultimately eliminated. We must examine all combinations of rules with a possibly matching active cut. We illustrate a few representative cases below.

Let us consider first the interaction between $(\langle\mathsf{a}\rangle\text{R})$ and $(\langle\mathsf{a}\rangle\text{L})$. We adapt the strategy presented in [35]. Suppose that in a derivation we encounter an application of (Cut) of minimal height of the form:

$$\dfrac{\dfrac{@_i\langle\mathsf{a}\rangle j, \Gamma \vdash^n \Delta, @_i\langle\mathsf{a}\rangle\varphi, @_j\varphi}{@_i\langle\mathsf{a}\rangle j, \Gamma \vdash \Delta, @_i\langle\mathsf{a}\rangle\varphi}\,(\langle\mathsf{a}\rangle\text{R}) \qquad \dfrac{@_i\langle\mathsf{a}\rangle j, @_j\varphi, \Gamma' \vdash^m \Delta'}{@_i\langle\mathsf{a}\rangle\varphi, \Gamma' \vdash \Delta'}\,(\langle\mathsf{a}\rangle\text{L})}{@_i\langle\mathsf{a}\rangle j, \Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\text{Cut})$$

The exhibited (Cut) has complexity $(\text{size}(@_i\langle\mathsf{a}\rangle\varphi), (n+1)+(m+1))$. We proceed to transform the derivation into a new one that reduces the complexity and moves us closer to a cut-free derivation.

$$\dfrac{\dfrac{@_i\langle\mathsf{a}\rangle j, \Gamma, \vdash^n \Delta, @_j\varphi, @_i\langle\mathsf{a}\rangle\varphi \qquad \dfrac{@_i\langle\mathsf{a}\rangle j, @_j\varphi, \Gamma' \vdash^m \Delta'}{@_i\langle\mathsf{a}\rangle\varphi, \Gamma' \vdash \Delta'}\,(\langle\mathsf{a}\rangle\text{L})}{@_i\langle\mathsf{a}\rangle j, \Gamma, \Gamma' \vdash \Delta, \Delta', @_j\varphi}\,(\text{Cut}_1) \qquad @_j\varphi, @_i\langle\mathsf{a}\rangle j, \Gamma' \vdash^m \Delta'}{@_i\langle\mathsf{a}\rangle j, \Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\text{Cut}_2)$$

In the transformed derivation, there are two applications of (Cut), labeled (Cut_1) and (Cut_2). We reason as follows, (Cut_1), is the only (Cut) in its subderivation and has complexity $(\text{size}(@_i\langle\mathsf{a}\rangle\varphi), n+(m+1))$. Therefore, by the inductive hypothesis, (Cut_1) can be eliminated resulting in a cut-free derivation of $@_i\langle\mathsf{a}\rangle j, \Gamma, \Gamma' \vdash^h \Delta, \Delta', @_j\varphi$, for some unknown h. We use this cut-free derivation as a building block to

construct the derivation:

$$\dfrac{@_i\langle \mathsf{a}\rangle j, \Gamma, \Gamma' \vdash^h \Delta, \Delta', @_j\varphi \qquad @_j\varphi, @_i\langle \mathsf{a}\rangle j, \Gamma' \vdash^m \Delta'}{@_i\langle \mathsf{a}\rangle j, \Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\mathrm{Cut}_2)$$

We can now see that (Cut_2) can also be eliminated. It has complexity $(\mathrm{size}(@_j\varphi), h+m)$ and it is the only (Cut) application in the derivation. Since $\mathrm{size}(@_j\varphi) < \mathrm{size}(@_i\langle \mathsf{a}\rangle\varphi)$, we can apply the inductive hypothesis, and obtain a fully cut-free derivation of $@_i\langle \mathsf{a}\rangle j, \Gamma, \Gamma' \vdash \Delta, \Delta'$.

Let us now consider a case involving data comparisons: the interaction between ($\langle\blacktriangle\rangle$R) and ($\langle\blacktriangle\rangle$L). Namely, suppose that in a derivation we encounter an application of (Cut) of minimal height of the form:

$$\dfrac{\dfrac{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \Gamma \vdash^n \Delta, @_i\langle\alpha \blacktriangle \beta\rangle, \langle j{:}\blacktriangle k{:}\rangle}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \Gamma \vdash \Delta, @_i\langle\alpha \blacktriangle \beta\rangle}\,(\langle\blacktriangle\rangle\mathrm{R}) \qquad \dfrac{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\blacktriangle k{:}\rangle, \Gamma' \vdash^m \Delta'}{@_i\langle\alpha \blacktriangle \beta\rangle, \Gamma' \vdash \Delta'}\,(\langle\blacktriangle\rangle\mathrm{L})}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\mathrm{Cut})$$

We transform this derivation into:

$$\dfrac{\dfrac{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \Gamma \vdash^n \Delta, \langle j{:}\blacktriangle k{:}\rangle, @_i\langle\alpha \blacktriangle \beta\rangle \qquad @_i\langle\alpha \blacktriangle \beta\rangle, \Gamma' \vdash \Delta'}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \Gamma, \Gamma' \vdash \Delta, \Delta', \langle j{:}\blacktriangle k{:}\rangle}\,(\mathrm{Cut}_1) \qquad \dfrac{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \langle j{:}\blacktriangle k{:}\rangle, \Gamma' \vdash^m \Delta'}{\langle j{:}\blacktriangle k{:}\rangle, @_i\langle\alpha\rangle j, @_i\langle\beta\rangle j, \Gamma' \vdash^m \Delta'}\,(\langle\blacktriangle\rangle\mathrm{L})}{@_i\langle\alpha\rangle j, @_i\langle\beta\rangle k, \Gamma, \Gamma' \vdash \Delta, \Delta'}\,(\mathrm{Cut}_2)$$

The argument is similar, the original use of (Cut) has complexity $(\mathrm{size}(@_i\langle\alpha \blacktriangle \beta\rangle), (n+1)+(m+1))$. When we transform the derivation, we introduce two new applications of (Cut), labeled (Cut_1) and (Cut_2). (Cut_1) has complexity $(\mathrm{size}(@_i\langle\alpha \blacktriangle \beta\rangle), n+(m+1))$ and can be eliminated. Then, (Cut_2), can be eliminated from the resulting derivation as it involves a simpler active cut expression.

For the particular to **G** case, let us consider the interaction between the rules ($\langle\mathsf{a}\rangle$R) and ($\langle\blacktriangle\rangle$R). Namely, suppose that in a derivation we encounter an application of (Cut) of minimal height of the form:

$$\dfrac{\dfrac{@_i\langle\mathsf{a}\rangle j, \Gamma \vdash^n \Delta, @_i\langle\mathsf{a}\rangle a, @_j a}{@_i\langle\mathsf{a}\rangle j, \Gamma \vdash \Delta, @_i\langle\mathsf{a}\rangle a}\,(\langle\mathsf{a}\rangle\mathrm{R}) \qquad \dfrac{@_i\langle\mathsf{a}\rangle a, @_i\langle\beta\rangle b, \Gamma' \vdash^m \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle, \langle a{:}\blacktriangle b{:}\rangle}{@_i\langle\mathsf{a}\rangle a, @_i\langle\beta\rangle b, \Gamma' \vdash \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle}\,(\langle\blacktriangle\rangle\mathrm{R})}{@_i\langle\mathsf{a}\rangle j, @_i\langle\beta\rangle b, \Gamma, \Gamma' \vdash \Delta, \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle}\,(\mathrm{Cut})$$

Unlike the other cut-elimination cases we have examined, this particular case involves two right rules applied in the premisses of (Cut), rather than a mix of left and right rules. This synchronization of right rules is unusual, but it does not pose a problem. We can still transform the derivation to eliminate (Cut).

$$\mathscr{D} = @_i\langle\mathsf{a}\rangle j, \Gamma \vdash^n \Delta, @_j a, @_i\langle\mathsf{a}\rangle a$$

$$\dfrac{\dfrac{\mathscr{D} \qquad \dfrac{@_i\langle\mathsf{a}\rangle a, @_i\langle\beta\rangle b, \Gamma' \vdash^m \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle, \langle a{:}\blacktriangle b{:}\rangle}{@_i\langle\mathsf{a}\rangle a, @_i\langle\beta\rangle b, \Gamma' \vdash \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle}\,(\langle\blacktriangle\rangle\mathrm{R})}{@_i\langle\mathsf{a}\rangle j, @_i\langle\beta\rangle b, \Gamma, \Gamma' \vdash \Delta, \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle, @_j a}\,(\mathrm{Cut}_1) \qquad \dfrac{\dfrac{\dfrac{@_i\langle\mathsf{a}\rangle a, @_i\langle\beta\rangle b, \Gamma' \vdash^m \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle, \langle a{:}\blacktriangle b{:}\rangle}{@_i\langle\mathsf{a}\rangle a, @_i\langle\beta\rangle b, \Gamma' \vdash \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle}\,(\langle\blacktriangle\rangle\mathrm{R})}{@_i\langle\mathsf{a}\rangle j, @_i\langle\mathsf{a}\rangle a, @_i\langle\beta\rangle b, \Gamma' \vdash \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle}\,(\mathrm{WL})}{@_j a, @_i\langle\mathsf{a}\rangle j, @_i\langle\mathsf{a}\rangle a, @_i\langle\beta\rangle b, \Gamma' \vdash \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle}\,(\mathrm{WL})}{@_j a, @_i\langle\mathsf{a}\rangle j, @_i\langle\beta\rangle b, \Gamma' \vdash \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle}\,(\mathrm{S}_2)}{@_i\langle\mathsf{a}\rangle j, @_i\langle\beta\rangle b, \Gamma, \Gamma' \vdash \Delta, \Delta', @_i\langle\mathsf{a} \blacktriangle \beta\rangle}\,(\mathrm{Cut}_2)$$

Just as in earlier cases, the original (Cut) is replaced by two new applications: (Cut_1), which has strictly smaller cut height, and (Cut_2), which involves a simpler active cut. Both of these are less complex than the original in the lexicographic sense, so we can apply the inductive hypothesis to eliminate them. This shows that even when both rules in the (Cut) are right rules, the same general strategy still works. The remaining cases of interaction of rules in cut-elimination can be found in [11]. □

5 A Sequent System for Hybrid Logic

It is helpful to view $\mathrm{HXPath}_{\mathrm{D}}$ as a modular extension of the Basic Hybrid Logic $\mathscr{H}(@)$ with additional constructs for data comparison. This naturally raises the question: can we recover a sequent calculus $\mathbf{G}'$ for $\mathscr{H}(@)$ by simply removing the rules related to data comparisons from $\mathbf{G}$? In this section, we make this correspondence precise by showing that the resulting fragment of $\mathbf{G}$ faithfully simulates the sequent calculus for $\mathscr{H}(@)$ presented in [17].

Definition 17. *A* formula of $\mathscr{H}(@)$ *is a node expression without data comparisons, and where diamonds are only of the form* $\langle \mathsf{a} \rangle \varphi$. *A* hybrid model for $\mathscr{H}(@)$ *is obtained by dropping the* $\{\approx_{\mathsf{c}}\}_{\mathsf{c} \in \mathsf{Cmp}}$ *component from the hybrid data models in Def. 2. The semantics of formulas on hybrid models is as in Def. 3. Let* $\mathbf{G}'$ *be obtained from* $\mathbf{G}$ *by restricting sequents to formulas in* $\mathscr{H}(@)$ *of the form* $@_i \varphi$, *and dropping all rules involving data comparisons.*

The calculus $\mathbf{G}'$ simulates the sequent calculus for $\mathscr{H}(@)$ in [17], which we refer to as $\mathbf{G}_{\mathscr{H}(@)}$

Lemma 18. *Every provable sequent in* $\mathbf{G}_{\mathscr{H}(@)}$ *is also provable in* $\mathbf{G}'$.

Proof. We establish this result by showing that every rule in $\mathbf{G}_{\mathscr{H}(@)}$ is a derived rule in $\mathbf{G}'$. The axioms in $\mathbf{G}_{\mathscr{H}(@)}$ align exactly with those in $\mathbf{G}'$, establishing a one-to-one correspondence. Additionally, ($\to$L), ($\to$R), (@L), and (@R) are present in both calculi. The rule (Ref) from $\mathbf{G}_{\mathscr{H}(@)}$ is also present in $\mathbf{G}'$ under the name (@T). The rules ($\wedge$L) and ($\wedge$R) are included as basic rules in $\mathbf{G}_{\mathscr{H}(@)}$, in contrast, they are derived in $\mathbf{G}'$. Similarly, the rules ([a]L) and ([a]R), and (Nom_1) and (Nom_2), listed below, which are primitive in $\mathbf{G}_{\mathscr{H}(@)}$, are also derivable in $\mathbf{G}'$. This shows that every derivation in $\mathbf{G}_{\mathscr{H}(@)}$ can be simulated in $\mathbf{G}'$, completing the correspondence between the two systems. Details can be found in [11].

$$\frac{\Gamma \vdash \Delta, @_i j \quad \Gamma \vdash \Delta, @_i \varphi}{\Gamma \vdash \Delta, @_j \varphi}\,(\mathrm{Nom}_1) \qquad \frac{\Gamma \vdash \Delta, @_i j \quad \Gamma \vdash \Delta, @_i \langle \mathsf{a} \rangle k \quad @_j \langle \mathsf{a} \rangle k, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\,(\mathrm{Nom}_2)$$

$$\frac{\Gamma \vdash \Delta, @_i \langle \mathsf{a} \rangle j \quad @_j \varphi, \Gamma \vdash \Delta}{@_i [\mathsf{a}] \varphi, \Gamma \vdash \Delta}\,([\mathsf{a}]\mathrm{L}_1) \qquad \frac{@_i \langle \mathsf{a} \rangle j, \Gamma \vdash \Delta, @_j \varphi}{\Gamma \vdash \Delta, @_i [\mathsf{a}] \varphi}\,([\mathsf{a}]\mathrm{R})\ j \text{ is new.}$$

As a concrete illustration of the correspondence between $\mathbf{G}_{\mathscr{H}(@)}$ and $\mathbf{G}$, we show how the rule Nom_2 from $\mathbf{G}_{\mathscr{H}(@)}$ can be obtained as a derived rule within $\mathbf{G}$.

$$\dfrac{\Gamma \vdash \Delta, @_i j \qquad \dfrac{\dfrac{\Gamma \vdash \Delta, @_i \langle \mathsf{a} \rangle k}{@_i j, \Gamma \vdash \Delta, @_i \langle \mathsf{a} \rangle k}\,(\mathrm{WL}) \qquad \dfrac{\dfrac{@_j \langle \mathsf{a} \rangle k, \Gamma \vdash \Delta}{@_j \langle \mathsf{a} \rangle k, @_i j, @_i \langle \mathsf{a} \rangle k, \Gamma \vdash \Delta}\,(\mathrm{WL})}{@_i \langle \mathsf{a} \rangle k, @_i j, \Gamma \vdash \Delta}\,(\mathrm{S}_1)}{@_i j, \Gamma \vdash \Delta}\,(\mathrm{Cut})}{\Gamma \vdash \Delta}\,(\mathrm{Cut})$$

We can see from the derivation how (Nom_1) is directly related to (S_1) in $\mathbf{G}$. □

Theorem 19 (Soundness and Completeness)**.** *A sequent in* $\mathscr{H}(@)$ *is valid in all hybrid models iff it is provable in* $\mathbf{G}'$.

In conclusion, the system $\mathbf{G}_{\mathscr{H}(@)}$ in [17] is derived from a natural deduction system for $\mathscr{H}(@)$ presented in the same work. To reflect the structure and constraints of such a natural deduction system, $\mathbf{G}_{\mathscr{H}(@)}$ does not include weakening rules and, more importantly, is designed to be cut-free. This comes with a trade-off: the rules (Nom_1), (Nom_2), and ([a]L) in $\mathbf{G}_{\mathscr{H}(@)}$ are not invertible in $\mathbf{G}_{\mathscr{H}(@)}$. In contrast, as a subsystem of $\mathbf{G}$, $\mathbf{G}'$ is fully invertible and enjoys cut elimination. We bring attention also to the fact

that, while (Cut) is admissible in $\mathbf{G}_{\mathscr{H}(@)}$, its use in a derivation cannot be systematically removed. Thus, by supporting both admissibility and effective elimination of (Cut), $\mathbf{G}'$ offers a more robust and analytically tractable framework for $\mathscr{H}(@)$.

6 Final Comments

In this article, we introduce the Gentzen-style sequent calculus **G** for the logic Hybrid XPath with Data (HXPath_D). HXPath_D is a hybrid modal logic that captures not only the navigational core of XPath but also data comparisons, node labels (keys), and key-based navigation operators. In **G**, we provide rules for handling complex path expressions (composition and tests), as well as mechanisms for data comparisons (equality and inequality), nominals, and satisfiability operators.

We establish the completeness of **G** by leveraging the completeness of the Hilbert-style calculus **H** for HXPath_D introduced in [10]. We also show that every rule in **G** is invertible, a crucial property in proof search. The system **G** includes weakening rules (WL) and (WR), and the (Cut) rule. (WL) and (WR) are never needed in derivations (they are only used to simplify sequents and eliminate irrelevant formulas at a given point in a proof). Moreover, we showed that (Cut) is also irrelevant for completeness, and we provide a cut elimination algorithm. Finally, we discuss how to define a subcalculus of **G**, which is a sound and complete sequent calculus for the Basic Hybrid Logic $\mathscr{H}(@)$, inheriting rule invertibility and cut elimination. This system improves on previously known sequent calculi for $\mathscr{H}(@)$ (see [17]).

Similarly to the approach in [17], we can extend **G** by incorporating pure axioms and existential saturation rules. A general result can be proved, showing that these extended calculi are sound and complete with respect to a large family of frame classes (see [11] for further details). There is a cost though: rule invertibility and cut-elimination are not guaranteed in these extensions. Future work will explore whether these properties can be preserved under certain reasonable conditions.

Many aspects of XPath have been extensively investigated from a logical perspective, but their proof-theoretical properties have remained largely unexplored. To our knowledge, the only exception is the sequent calculus for DataGL from [14]. In brief, DataGL can be seen as a formalization of a highly restricted fragment of XPath in which data comparisons are allowed only between the evaluation point and its strict descendants in a data tree. In comparison, **G** provides much more expressivity.

The proof theory of modal logics has always been challenging. One reason for this is that modal languages combine a simple propositional syntax with a rich relational semantics. As a result, proof-theoretical approaches that rely solely on the *form* of logical expressions struggle to capture their full expressive power. Various proof-theoretical techniques——such as display calculi, hypersequents, deep inference, labeled systems, and hybridization——have been explored to establish a well-rounded proof theory for modal logics. Data-aware modal logics extend the expressive power of traditional modal languages even further, incorporating mechanisms to handle pairs of complex paths in graphs and data comparisons. Our results indicate that hybridization techniques (i.e., internalizing the use of labels via nominals and satisfiability operators) can provide the basis for a solid proof theory for modal logics with data-comparison operators.

There are several open lines for future research. In particular, we would like to prove the termination of our calculus and define an optimal proof search strategy. Moreover, we wish to extend the calculus to more expressive fragments—such as those involving sibling relations, transitive closure navigation, or novel forms of data comparison, and to study proof theoretical aspects of an intuitionistic variant of HXPath_D (see [8]).

References

[1] S. Abriola, P. Barceló, D. Figueira & S. Figueira (2016): *Bisimulations on Data Graphs*. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR*, pp. 309–318.

[2] S. Abriola, P. Barceló, D. Figueira & S. Figueira (2018): *Bisimulations on Data Graphs*. *Journal of Artificial Intelligence Research* 61, pp. 171–213, doi:10.1613/jair.5637.

[3] S. Abriola, M. Descotte, R. Fervari & S. Figueira (2017): *Axiomatizations for downward XPath on data trees*. *Journal of Computer and System Sciences* 89, pp. 209–245, doi:10.1016/j.jcss.2017.05.008.

[4] S. Abriola, M. Descotte & S. Figueira (2014): *Definability for Downward and Vertical XPath on Data Trees*. In: *21th Workshop on Logic, Language, Information and Computation, LNCS* 6642, pp. 20–34, doi:10.1007/978-3-662-44145-9_2.

[5] S. Abriola, M. Descotte & S. Figueira (2017): *Model theory of XPath on data trees. Part II: Binary bisimulation and definability*. *Information and Computation* 255, pp. 195–223, doi:10.1016/J.IC.2017.01.002.

[6] S. Abriola, S. Figueira & N. González (2024): *Axiomatization of XPath with general data comparison*. *Journal of Applied Non-Classical Logics*, pp. 1–20.

[7] R. Angles, M. Arenas, P. Barceló, P. A. Boncz, G. H. L. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. F. Sequeda, O. van Rest & H. Voigt (2018): *G-CORE: A Core for Future Graph Query Languages*. In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018*, ACM, pp. 1421–1432, doi:10.1145/3183713.3190654.

[8] C. Areces, V. Cassano, D. Dutto & R. Fervari (2023): *Data Graphs with Incomplete Information (and a Way to Complete Them)*. In: *18th European Conference on Logics in Artificial Intelligence (JELIA 2023), LNCS* 14281, Springer, pp. 729–744, doi:10.1007/978-3-031-43619-2_49.

[9] C. Areces & B. ten Cate (2006): *Hybrid Logics*. In: *Handbook of Modal Logic*, Elsevier, pp. 821–868.

[10] C. Areces & R. Fervari (2021): *Axiomatizing Hybrid XPath with Data*. *Logical Methods in Computer Science* 17(3), doi:10.46298/LMCS-17(3:5)2021.

[11] Carlos Areces, Valentin Cassano, Danae Dutto & Raul Fervari (2025): *Sequent Calculi for Data-Aware Modal Logics: Compendium of Proofs*. arXiv:2505.17240.

[12] A. Avron (1996): *The method of hypersequents in the proof theory of propositional nonclassical logics*. In W. Hodges, M. Hyland, C. Steinhorn & J. Truss, editors: *Logic: from foundations to applications*, Oxford Science Publications, pp. 1–32.

[13] D. Baelde, A. Lick & S. Schmitz (2019): *Decidable XPath Fragments in the Real World*. In: *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019*, ACM, pp. 285–302, doi:10.1145/3294052.3319685.

[14] D. Baelde, S. Lunel & S. Schmitz (2016): *A Sequent Calculus for a Modal Logic on Finite Data Trees*. In: *25th EACSL Annual Conference on Computer Science Logic (CSL 2016), LIPIcs* 62, Schloss Dagstuhl, pp. 32:1–32:16, doi:10.4230/LIPICS.CSL.2016.32.

[15] M. Benedikt & C. Koch (2008): *XPath leashed*. *ACM Computing Surveys* 41(1), doi:10.1145/1456650.1456653.

[16] M. Bojańczyk, A. Muscholl, T. Schwentick & L. Segoufin (2009): *Two-variable logic on data trees and XML reasoning*. *Journal of the ACM* 56(3), doi:10.1145/1516512.1516515.

[17] T. Braüner (2011): *Hybrid Logic and its Proof-Theory*. *Applied Logics Series* 37, Springer.

[18] P. Buneman (1997): *Semistructured data*. In: *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '97, Association for Computing Machinery, New York, NY, USA, p. 117–121, doi:10.1145/263661.263675.

[19] B. ten Cate, T. Litak & M. Marx (2010): *Complete axiomatizations for XPath fragments*. *Journal of Applied Logic* 8(2), pp. 153–172, doi:10.1016/j.jal.2009.09.002.

[20] B. ten Cate & M. Marx (2009): *Axiomatizing the Logical Core of XPath 2.0*. Theory of Computing Systems 44(4), pp. 561–589, doi:10.1007/11965893_10.

[21] D. Figueira (2010): *Reasoning on Words and Trees with Data*. PhD thesis, Laboratoire Spécification et Vérification, ENS Cachan, France.

[22] D. Figueira (2012): *Decidability of Downward XPath*. ACM Transactions on Computational Logic 13(4), p. 34, doi:10.1145/2362355.2362362.

[23] D. Figueira (2018): *Satisfiability of XPath on data trees*. ACM SIGLOG News 5(2), pp. 4–16, doi:10.1145/3212019.3212021.

[24] D. Figueira, S. Figueira & C. Areces (2014): *Basic Model Theory of XPath on Data Trees*. In: International Conference on Database Theory, pp. 50–60, doi:10.5441/002/ICDT.2014.09.

[25] D. Figueira, S. Figueira & C. Areces (2015): *Model Theory of XPath on Data Trees. Part I: Bisimulation and Characterization*. Journal of Artificial Intelligence Research 53, pp. 271–314, doi:10.1613/JAIR.4658.

[26] D. Figueira & L. Segoufin (2011): *Bottom-up automata on data trees and vertical XPath*. In: 28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011), pp. 93–104, doi:10.4230/LIPICS.STACS.2011.93.

[27] N. Francis, A. Gheerbrant, P. Guagliardo, L. Libkin, V. Marsault, W. Martens, F. Murlak, L. Peterfreund, A. Rogova & D. Vrgoc (2023): *GPC: A Pattern Calculus for Property Graphs*. In: 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, (PODS'23), ACM, pp. 241–250, doi:10.1145/3584372.3588662.

[28] N. Francis, A. Gheerbrant, P. Guagliardo, L. Libkin, V. Marsault, W. Martens, F. Murlak, L. Peterfreund, A. Rogova & D. Vrgoc (2023): *A Researcher's Digest of GQL*. In: 26th International Conference on Database Theory, LIPIcs 255, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 1:1–1:22, doi:10.4230/LIPICS.ICDT.2023.1.

[29] G. Gottlob, C. Koch & R. Pichler (2005): *Efficient algorithms for processing XPath queries*. ACM Transactions on Database Systems 30(2), pp. 444–491, doi:10.1145/1071610.1071614.

[30] (2024): *ISO/IEC 39075. 2024. Information technology - Database languages - GQL*. Standard, International Organization for Standardization, Geneva, CH.

[31] L. Libkin, W. Martens & D. Vrgoč (2016): *Querying Graphs with Data*. Journal of the ACM 63(2), pp. 14:1–14:53, doi:10.1145/2850413.

[32] L. Libkin & D. Vrgoč (2012): *Regular path queries on graphs with data*. In: International Conference on Database Theory, ACM, pp. 74–85, doi:10.1145/2274576.2274585.

[33] S. Negri (2005): *Proof analysis in modal logic*. Journal of Philosophical Logic 34, pp. 507–544.

[34] S. Negri (2011): *Proof Theory for Modal Logic*. Philosophy Compass 6(8), pp. 523–538.

[35] S. Negri & J. von Plato (2014): *Proof Analysis: A Contribution to Hilbert's Last Problem*. Cambridge University Press.

[36] I. Robinson, J. Webber & E. Eifrem (2013): *Graph Databases*. O'Reilly Media, Inc.

[37] C. Stewart & P. Stouppa (2004): *A Systematic Proof Theory for Several Modal Logics*. In R. Schmidt, I. Pratt-Hartmann, M. Reynolds & H. Wansing, editors: Advances in Modal Logic 5, 2004, King's College Publications, pp. 309–333.

[38] H. Wansing (2002): *Sequent systems for modal logics*. In Dov Gabbay & Franz Guenthner, editors: Handbook of Philosophical Logic, 2 edition, 8, Kluwer, pp. 61–145.

Characterization of Lattice Properties Within Modal Extensions

Alfredo R. Freire
Department of Philosophy
University of Brasilia
Brasilia, Brasil
alfredo.filho@unb.br

Manuel A. Martins
Department of Mathematics
University of Aveiro
Aveiro, Portugal
martins@ua.pt

This paper investigates the extension of lattice-based logics into modal languages. We observe that such extensions admit multiple approaches, as the interpretation of the necessity operator is not uniquely determined by the underlying lattice structure. The most natural interpretation defines necessity as the meet of the truth values of a formula across all accessible worlds—an approach we refer to as the *normal interpretation*. We examine the logical properties that emerge under this and other interpretations, including the conditions under which the resulting modal logic satisfies the axiom K and other common modal validities. Furthermore, we consider cases in which necessity is attributed exclusively to formulas that hold in all accessible worlds.

1 Introduction

Introduced by Kripke [Kri59], the classical modal semantics consists of a set of worlds, an accessibility relation between worlds, and valuations of propositional variables over the two-valued Boolean algebra B_2. However, instead of using B_2 as the basis for valuations and non-modal connectives, one may start from an alternative lattice semantics such as the Łukasiewicz algebra $[0,1]$, the four-valued Belnap lattice or the Boolean algebras of n with $n > 2$. In this paper, we study modal extensions of lattice semantics, how some modal sentences can impose constraints on the lattices.

Lattices can be used as semantics for an important family of logics including classical, intuitionistic, fuzzy, relevant, and paraconsistent systems. They interpret disjunction and conjunction in a familiar way and structure logical values in a partially ordered set. We shall briefly discuss possibilities for building modal extensions from a given lattice semantics. However, as the ordered structure of lattices suggests natural choices for interpreting the necessity operator – namely, the greatest lower bound of values in the accessed worlds – we adopt this interpretation for most of our results.

Our interest in this kind of modal logic is rooted in the problem of meaningfully interpreting, within a given logic, the truth values of another logic. This is fundamentally important if we want to address the **counterlogical** statements.[1] Moreover, this issue has appeared in applications to computer systems where states operate using different algebraic properties (see [MMH19; BMC14; DS07]). In this context, a lattice L containing lattice semantics L_1 and L_2 as sublattices provides us with a common order in which logics based on L_1 and L_2 can meaningfully communicate. This framework was introduced in [FM25], and a general mathematical treatment of these structures is worth developing. Here, we limit our work to

[1]The issue of counterlogical statements has been treated in recent literature by Nolan in [Nol13], Berto and Jago in the recent book [BJ19] and Kocurek and Jerzak in [KJ21]. The general approach is to consider, in opposition to **normal** worlds, the **impossible** worlds where formulas do not always receive values by algebraic operations from the values of propositional variables. This, we believe, makes the study of logical possibilities very restrictive. For example, one cannot access an intuitionistic world from the point of view of a classical world, and one must choose a logic that should be considered normal.

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. 26–36, doi:10.4204/EPTCS.430.3

structures where every world operates in a single lattice semantics. This is a fundamental and required development if we want to properly generalize the treatment of many logics across many worlds proposed in [FM25] (see also [FMM24] for an institutional formulation of Many Logics Modal Logic).

After a short section on lattice semantics background and a section on modal extension of lattice semantics, we study the relation between general modal validities and lattice properties. We will first observe that factorization of $\Box$ over disjunction is fundamentally related to the order relations in the set of designated values. In addition, we will observe some general conditions connected to the validity of axiom K. The paper concludes with some lines for future research.

2 Background on lattice semantics

Before going into the content of our paper, let us briefly introduce some definitions and basic concepts that we will use throughout the paper.

A **partially ordered set** is an ordered pair $(P,\leq)$ in which P is a set and $\leq$ is a binary relation on P reflexive, anti-symmetric and transitive. Two elements $a,b \in P$ are said to be **comparable to** $\leq$ if $a \leq b$ or $b \leq a$. The order $(P,\leq)$ will be called **linearly ordered** if any two elements can be compared using $\leq$.

A *lattice* is a partially ordered nonempty set $(L,\leq)$ in which each pair of elements a, b has a join (denoted by $a+b$) and a meet (denoted by $a.b$). In some contexts, it is useful to consider a unary operation on L called **complementation** and denoted by the symbol $-$[2]. A lattice L with complementation is **anti-monotone** when for every $a,b \in L$, if $a \leq b$, then $-b \leq -a$. Let $\bigwedge S$ denote (if it exists) the greatest lower bound of the values in a set S, we say that the lattice L has **down-distribution** when, for all $A \subseteq L$ and $B \subseteq L$, we have $\bigwedge(A+B) = \bigwedge A + \bigwedge B$. In general, we will use $a \supset b$ to refer to $-a+b$, although some alternatives will also be considered.

In this paper, a **set of designated elements** of a partially ordered set $\langle L,\leq\rangle$ is a subset A of L that is closed upward[3], that is, for any $a,b \in L$, $a \leq b$ and $a \in A$ imply $b \in A$. For a lattice L, we say that A is **implicative** in L when

$$a \leq b \Rightarrow a \supset b \in A.$$

A set of designated elements of L is called a **filter** if it is closed under meet operation. A filter is called an **ultrafilter** if it is maximal with respect to inclusion.

A **matrix** is a pair $\langle L,D\rangle$, where L is a lattice and D is a set of designated values. If D is a filter, $\langle L,D\rangle$ is called a **filter-matrix**.

Let $\mathcal{L} = \{Var,\wedge,\vee,\neg\}$ be a propositional language and Var be a set of propositional variables; $Form(\mathcal{L}, Var)$, the set of propositional formulas of $\mathcal{L}$, is the free $\mathcal{L}$-algebra over Var. A **valuation** is a function of Var in L. It is well known that it can be uniquely extended to a homomorphism, namely the map $\bar{v}: Form(\mathcal{L},Var) \to L$ defined in the following way

1. $\bar{v}(p) = v(p)$ for every variable $p \in Var$;
2. $\bar{v}(\varphi \vee \psi) = \bar{v}(\varphi) + \bar{v}(\psi)$;
3. $\bar{v}(\varphi \wedge \psi) = \bar{v}(\varphi).\bar{v}(\psi)$;

[2]The name complementation suggests that it will be used in the interpretation of negation and suggests that it will have the Boolean properties $-a+a = 1$ and $-a.a = 0$. Although we intend to use complementation to interpret negation, we shall not commit ourselves upfront to any property for the operation of complementation.

[3]This, of course, diminishes the generality of the set of designated values being considered. However, it is rare to see such sets employed in defining the concept of validity within a logical framework.

4. $\bar{v}(\neg\varphi) = -\bar{v}(\varphi)$.

Since $\bar{v}$ is uniquely determined by v we simply write v. A **substitution** is an automorphism in the formula algebra $Form(\mathcal{L}, Var)$. Given a matrix $M = (L, D)$, a valuation v and a formula φ, we write $v \vDash_M \varphi$ whenever $v(\varphi) \in D$. The **consequence relation over** M is the relation $\vDash_M \subseteq P(Form(\mathcal{L}, Var)) \times Form(\mathcal{L}, Var)$ defined by

$$\Gamma \vDash_M \varphi \text{ iff for all valuations } v,\ v(\Gamma) \subseteq D \text{ implies } v(\varphi) \in D.$$

When it is clear from the context, we omit explicit references to M in $\vDash_M$. The relation $\vDash_M$ satisfies Tarski's conditions.

3 Modal extension of lattice based logics

We restrict our investigation to extensions of logics that have a lattice semantics as suggested in [Fit91; Pri08b; OL12; FM25]. This restriction will allow us to produce a broader study of the phenomena of modal structures since it produces common meet and join operations shared by most familiar logics.

As we shall see shortly, there are many alternatives for interpreting the validity of $\Box\varphi$ in a world. If we consider a model M where every world operates in a three-valued lattice $\{0, 0.5, 1\}$ with distinguished values $\{0.5, 1\}$, one could say that the value of the formula $\Box\psi$ is 1 whenever all accessible worlds validate ψ (i.e., ψ has a designated value in all accessible worlds). We could, however, attribute the value 0.5 or find other more articulated ways of attributing the truth value for the modal formula. We shall call a modal valuation *regular* when it implies that "necessity means true in all accessible worlds" (NAW). More precisely, let us define a general modal extension of these logics:

Definition 1. *Let $\mathcal{L}$ be a propositional language, and Var be the set of propositional variables of $\mathcal{L}$. For a kripke frame*[4] *$F = \langle W, R\rangle$ and a lattice A , we say that v is an A-**valuation** of F in $\mathcal{L}$ when v is a function from $W \times Var$ to A.*

The function $v_w : Var \longrightarrow A$ is an A-valuation such that $v_w(X) = v(w, X)$ for every variable $X \in Var$.

Definition 2. *Let $F = \langle W, R\rangle$ be a Kripke frame and v an A-valuation for a lattice A, we define $M = \langle W, R, v\rangle$ to be a A-**model** with frame F.*

Definition 3. *For a lattice A with designated values D and corresponding logic semantics $\vDash_{A,D}$, we say that $\Vdash$ is a **modal extension** of $\vDash_{A,D}$ when for every A-model $M = \langle W, R, v\rangle$ and every world $w \in W$ we have*

1. *for every formula φ without modal operator and $w \in W$,*

$$w \Vdash \varphi \Longleftrightarrow v_w \vDash_{A,D} \varphi;$$

*We will say that the modal extension $\Vdash$ is **regular** when additionally*

2. *for every formula ψ and every $w \in W$,*

$$w \Vdash \Box\psi \Longleftrightarrow \text{for every } w' \in W \text{ with } wRw', w' \Vdash_{A,D} \psi$$

Example 1. *Let $\mathbf{B}_2$ be the Boolean algebra with two elements 0 and 1. Consider the consequence relation over the matrix $(\mathbf{B}_2, \{1\})$ (i.e., $\vDash_{B_2,\{1\}}$).*

[4] In modal logic a **Kripke frame** F is a pair= $\langle W, R\rangle$, where W is a set (of worlds) and $R \subseteq W^2$ (the accessibility relation).

The modal extension of $\models_{B_2,\{1\}}$ *where*

$$w \Vdash \Box\psi := \bigwedge\{w' \Vdash \psi | wRw'\}$$

coincides with the standard modal logic which is a regular modal extension of $\models_{B_2,\{1\}}$.

On the other hand, taken the same $\models_{B_2,\{1\}}$*, and considering the modal extension with*

$$w \Vdash \Box\psi := w \Vdash \psi$$

we obtain a non-regular modal extension.

We also have non-trivial examples of non-regular modal extensions.

Consider the logic $\models_{B,U}$*, where* B *is an infinite Boolean algebra and* U *is a non-principal ultrafilter on* B.

Define

$$w \Vdash \Box\psi := \bigwedge\{w' \Vdash \psi | wRw'\}.$$

Consider a model M *with a world* w_1 *that accesses all* w_b *for each* $b \in U$*. If we take* $v_{w_a}(p) = a$*, the value of* p *is designated in each world. i.e.* $w_a \Vdash p$*. However,* $w_1 \Vdash \Box p = \bigwedge U \notin U$ *as* U *is non-principal. Consequently,* $w_1 \nVdash \Box p$.

Naturally, if A has more than one distinguished value, it is easy to show that there are more than one regular modal extension. Moreover, whenever the lattice is complete, the most natural modal extension assigns to $\Box\varphi$ in a world w the conjunction of the values of φ in the accessible worlds. Thus, we define:

Definition 4. *Let* $\mathcal{L} = \{Var, \wedge, \vee, \neg\}$ *be a propositional language, and Form be the set of propositional formulas of* $\mathcal{L}$*. For a* ***complete*** *lattice* $A = \langle D, ., +, - \rangle$ *where* $.$ *is the usual meet operation,* $+$ *the usual join operation, and* $-$ *is any unary operation over* D*, consider the frame* $F = \langle W, R \rangle$*. We say that* $v : W \times Form \longrightarrow A$ *is a* ***normal modal valuation*** *when for every* $w \in W$

1. $v(w, X) \in A$ *for every variable* X *of* $\mathcal{L}$.
2. $v(w, \varphi \vee \psi) = v(w, \varphi) + v(w, \psi)$.
3. $v(w, \varphi \wedge \psi) = v(w, \varphi).v(w, \psi)$.
4. $v(w, \neg\varphi) = -v(w, \varphi)$.
5. $v(w, \Box\varphi) = \bigwedge\{v(w', \varphi) \mid wRw'\}$.

We generally use the notation $v_w(\varphi)$ *for* $v(w, \varphi)$.

Note that unless the lattice is complete, the definition of modal valuations will be partial. In cases where the base lattice is incomplete, there would be modal formulas that cannot have a truth value. This is because there will be a set of values for which the meet operation does not have a determinate value. We will return briefly to this topic later in this paper.

Definition 5. *Let* A *be a lattice with corresponding language* $\mathcal{L}$*,* $F \subset A$ *be a set of designated values, and* $M = \langle W, R, v \rangle$ *be any* A*-model where* v *is a normal modal valuation. For* $w \in W$ *and* φ *in* $\mathcal{L}$*, we say that* $w \Vdash_{A,F} \varphi$ *if, and only if,* $v_w(\varphi) \in F$*. We call* $\Vdash_{A,F}$ *the* ***normal modal extension*** *of the matrix* $H = \langle A, F \rangle$.

If there are other operations in the lattice (e.g., for implication or for a different kind of negation), a normal modal valuation applies the Definition 4 for the operations $\wedge, \vee, \neg$ and the corresponding operation for the additional operation. In this case, if $P : D^n \to D$ represents an operation in the lattice for the symbol p, the normal modal valuation will be such that $v(w, p(\varphi_1, \varphi_2, \ldots, \varphi_n)) = P(v(w, \varphi_1), v(w, \varphi_2), \ldots, v(w, \varphi_n))$.

Theorem 1. *For a complete lattice A and a set F of designate values in A, the A-normal modal valuation produces a unique regular modal extension of $\langle A, F\rangle$ if, and only if, $\bigwedge F \in F$ and F is a filter.*

Proof. Let us suppose F is a filter in A such that $\bigwedge F \in F$ and consider a A-model $M = \langle W, R, v\rangle$.

If we assume that there is a world $w \in W$ satisfying $\Box\varphi$, we obtain

$$\bigwedge\{v_{w'}(\varphi) \mid wRw'\} = f \in F$$

Therefore $v_{w'}(\varphi) \geq f$ for every w' accessible to w. Since F is a filter, $v_{w'}(\varphi) \in F$. This in turn means that each w' accessible to w satisfies φ.

If we assume that all w' accessible to w satisfies φ, we obtain $\{v_{w'}(\varphi) \mid wRw'\} \subseteq F$. It follows that $\bigwedge\{v_{w'}(\varphi) \mid wRw'\} \geq \bigwedge F$ and thus $v_w(\Box\varphi) \in F$ (i.e. $w \Vdash_A \Box\varphi$).

Now, let us assume that F is not a filter and there is $a \geq b \in F$ such that $a \notin F$. Then we consider a model with two worlds w and w' such that (i) wRw and wRw' and (ii) $v_w(C) = b$ and $v_{w'}(C) = a$ for a propositional variable C. We obtain $v_w(\Box C) = b$, and thus $w \Vdash_A C$. However, note that $w' \nVdash_A C$ since $a \notin F$.

If, on the other hand, $\bigwedge F \notin F$, then we may obtain a model in which there is a world w that accesses worlds w_f, $f \in F$, and such that for a propositional variable C we have $v_{w_f}(C) = f$ for every $f \in F$. Therefore, we find that every world accessed by w the formula C is valid, but that w does not validate $\Box C$.

□

Let us briefly consider the finite and infinite cases for lattices and how it relates to regular modal valuations. Indeed, every finite lattice is complete. Additionally, whenever the set of designated values F is a filter on the finite lattice A, we have $\bigwedge F \in F$. Hence, for a finite lattice, we can state theorem 1 simply as:

Corollary 1. *Let A be a finite grid, and F be a set of designated values in A. The normal extension $\Vdash_{A,F}$ is regular if, and only if, F is a filter.*

For infinite cases, we have different scenarios:

1. Consider the lattice D with domain in the real line interval $[0, 1]$ with the usual order and with negation $-x = 1 - x$. The normal modal valuation will produce a regular modal extension when the filter is $F_1 = [0.5, 1]$ and a non-regular modal extension when it is $F_2 =]0.5, 1]$. Note that $\bigwedge F_1 = 0.5 \in F_1$ and $\bigwedge F_2 = 0.5 \notin F_2$.
2. Moreover, consider the Boolean algebra B with the usual operations. Let U be an ultrafilter over B. Consequently, the logic operating in the normal modal extension will be classical logic. Note that every finite model will operate precisely in the same way as in classical Kripke finite structures. However, if the ultrafilter U is non-principal, then $\bigwedge U \notin U$. Consequently, the normal modal extension obtained will be regular if, and only if, the ultrafilter is principal.

This shows that we can produce natural instances where NAW fails as $\langle D, F_2\rangle$ and $\langle B, U\rangle$ are usual lattice semantics for fuzzy logics and classical logic, respectively.

4 Formulas characterizing lattice structures

Unless otherwise stated, we assume that implication $x \supset y$ is defined with the standard operation $-x + y$. However, it is important to note that this assumption is not favored in many important cases. Indeed,

the logic LP (see. [Pri08a, p. 142-160]) defines implication as $-x+y$ and this choice undermines *Modus Ponens*. However, a modification of implication can make *Modus Ponens* valid in LP's three-valued lattice. One such change is simply to attribute top value[5] whenever the consequent is bigger than the antecedent and attribute the value of the consequent otherwise:

$$v(X \supset Y) = \begin{cases} 1 & , v(X) \leq v(Y) \\ v(Y) & , \text{ otherwise} \end{cases} \tag{1}$$

This implication can be defined in any lattice that has a top value 1. Implication is defined in this way for many non-classical logics to guarantee that the resulting logic satisfies the deduction theorem. Observe further that having an implication as in Equation 1 can be very useful since in this case any set of designated values will be implicative.

Theorem 2. *If L is a lattice with top value* 1*, implication $\supset$ as in Equation 1 and F is a set of designated values in L, then F is implicative in L.*

Proof. Let a and b be values in L such that $a \leq b$. Then $a \supset b = 1$. Because F is upward closed, $1 \in F$ and so $(a \supset b) \in F$. □

Implicative filters preserve some important traditional modal validities in their normal modal extensions, as we see next. In short, we can show any implication by establishing that the antecedent is smaller than the consequent — which may be very convenient in lattice semantics. Recall that most notable modal properties are written in terms of implications such as axioms K, 4, 5 and so on.

Notation 1. *Let X and Y be subsets of the lattice L, we use $X+Y$ to refer to $\{x+y \mid x \in X \wedge y \in Y\}$.*

Lemma 1. *Let L be a lattice, F and G be subsets of L such that $\bigwedge F$, $\bigwedge G$ and $\bigwedge(F+G)$ exist, then*

$$\bigwedge F + \bigwedge G \leq \bigwedge(F+G)$$

Proof. Fix $a = \bigwedge(F+G)$, $b = \bigwedge F$ and $c = \bigwedge G$. Indeed, every $x \in (F+G)$ is bigger than b, since b is smaller than every member of F. Thus, from the maximality of a, $b \leq a$ and similarly $c \leq a$. Thence $b+c \leq a$ as desired. □

Lemma 2. *Let $L = \langle D, +, ., -, \supset \rangle$ be a complete lattice, and F be an implicative filter. Then for any L-model $M = \langle W, R, v \rangle$ and $w \in W$*

$$w \Vdash_{L,F} (\Box\varphi \vee \Box\psi) \to \Box(\varphi \vee \psi)$$

.

Proof. Let w be a world in a L-model $M = \langle W, R, v \rangle$. We note that

$$v_w(\Box\varphi \vee \Box\psi) = \bigwedge \underbrace{\{(v_{w'}(\varphi)) \mid wRw'\}}_{F} + \bigwedge \underbrace{\{(v_{w'}(\psi)) \mid wRw'\}}_{G}$$

So, from Lemma 1, we obtain that

$$v_w(\Box\varphi \vee \Box\psi) = \bigwedge F + \bigwedge G \leq \bigwedge(F+G)$$

But $\bigwedge(F+G) = \bigwedge\{(v_{w'}(\varphi) + v_{w'}(\psi)) \mid wRw'\}$. Thence, we have $v_w(\Box\varphi \vee \Box\psi) \leq v_w(\Box(\varphi \vee \psi))$. Since F is implicative, w satisfies the formula $(\Box\varphi \vee \Box\psi) \to \Box(\varphi \vee \psi)$. □

[5] If the lattice has a top value, we will generally refer to this value with 1 and, if it has a bottom value, we will generally refer to this value with 0.

We extend the property for models and frames using the usual strategy:

Definition 6. *Let A be a lattice, $F \subset A$ and $\Vdash$ be a modal extension of $\models_{A,F}$ in the language L. For a model $M = \langle W, R, v\rangle$, we will say that M satisfies the L-formula φ (i.e. $M \Vdash \varphi$) when, for every $w \in W$, $w \Vdash \varphi$.*

Definition 7. *Let A be a lattice, $F \subset A$ and $\Vdash$ be a modal extension of $\models_{A,F}$ in the language L. For a frame $\mathcal{F} = \langle W, R\rangle$, we will say that $\mathcal{F}$ satisfies the L-formula φ (i.e. $\mathcal{F} \Vdash \varphi$) when, for every A-model M with frame $\mathcal{F}$, $M \Vdash \varphi$.*

Theorem 3. *Let L be a complete lattice and F be a set of designated values in L. Then F is implicative if, and only if, every frame $\mathcal{F} = \langle W, R\rangle$ is such that $\mathcal{F} \Vdash_{L,F} (\Box\varphi \vee \Box\psi) \to \Box(\varphi \vee \psi)$.*

Proof. That every $\mathcal{F}$ satisfies $(\Box\varphi \vee \Box\psi) \to \Box(\varphi \vee \psi)$ under the designated values in F comes directly from the Lemma 2.

Now suppose F is not implicative. Then there are $a, b \in L$ such that $a \leq b$ and $a \supset b \notin F$. Consider a model M with at least the worlds w_1, w_2, w_3 such that $w_1 R w_2$ and $w_1 R w_3$ and valuation $v_{w_2}(\varphi) = a$, $v_{w_2}(\psi) = b$, $v_{w_3}(\varphi) = b$, $v_{w_3}(\psi) = a$. Then $v_{w_1}(\Box\varphi) = a$ and $v_{w_1}(\Box\psi) = a$; consequently $v_{w_1}(\Box\varphi \vee \Box\psi) = a$. On the other hand, $v_{w_1}(\Box(\varphi \vee \psi)) = b$. So $v_{w_1}((\Box\varphi \vee \Box\psi) \to \Box(\varphi \vee \psi)) \notin F$ as wanted. □

For dealing with axiom K, it will be instructive to consider different versions of implication. We will first consider the case that requires that implication satisfies a condition similar to that in Equation 1 and then we consider the case where $x \supset y$ is simply $-x + y$.

Definition 8. *Let L be a lattice and let $F \subset L$ be a set of designated values. Define implication ($\supset$) as any function $L \times L \to L$. We say that implication is **deductive** if for every $a, b \in A$*

1. *if $a \leq b$, then $b \leq (a \supset b) \in F$.*
2. *if $a \not\leq b$, then $(a \supset b) = b$.*

*We say that implication is **strictly deductive** if there is a top value 1 in L and $(a \supset b) = 1$ when $a \leq b$.*

Definition 9. *We say that a lattice L with filter F is **linear outside** F if every non-linearity of L is contained in F, i.e., if $x \in L$ is not comparable with some $y \in L$, then $x \in F$.*

Lemma 3. *Let M be a L-model, F be a filter in L and say that implication is deductive in L. If L is a linear lattice outside the filter, then every $w \in M$ satisfies ($\Vdash_{L,F}$) the axiom K, i.e. $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$.*

Proof. Suppose $w \in M$ is such that $w \nVdash_F \Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$. Because implication is deductive, $a = v_w(\Box(\varphi \to \psi)) \not\leq v_w(\Box\varphi \to \Box\psi) = c$ and $c \notin F$ for some values a and c in L. Subsequently, we have that $v_w(\Box\varphi) \not\leq v_w(\Box\psi)$, $v_w(\Box\psi) \notin F$ and $v_w(\Box\psi) = c$. Consider $A = \{v_x(\varphi \to \psi) \mid wRx\}$, $B = \{v_x(\varphi) \mid wRx\}$ and $C = \{v_x(\psi) \mid wRx\}$.

Call B_i and C_i the elements of B and C such that $v_x(\varphi) \not\leq v_x(\psi)$ and call B_j and C_j the elements of B and C such that $v_x(\varphi) \leq v_x(\psi)$. Recall that $\bigwedge C = c \notin F$ and L is linear outside F.

Let $c_i = \bigwedge C_i$ and $c_j = \bigwedge C_j$. Since $c = c_i.c_j$, $c \notin F$ and F is a filter, at least one of c_i and c_j is outside the filter. Because the lattice is linear outside the filter, we have $c_i = c$ or $c_j = c$. Suppose $c_i = c$. In this case, since every element of A is either in C_i or in the filter and c_i is smaller than every member of the filter, we conclude $\bigwedge A = a = c$. This in turn contradicts $v_w(\Box(\varphi \to \psi)) \not\leq v_w(\Box\varphi \to \Box\psi)$. So we have $c_j = c$. However, note that $\bigwedge B_j \leq \bigwedge C_j$ once every member of C_j has a smaller element in B_j. Trivially, $b \leq \bigwedge B_j$; so $b \leq c$, contradicting $v_w(\Box\varphi) \not\leq v_w(\Box\psi)$. □

Theorem 4. *Let L be a complete lattice, F be a non-empty set of designated values in L and implication be strictly deductive in L. Then L is linear outside F if, and only if, every frame $\mathcal{F} = \langle W, R\rangle$ is such that $\mathcal{F} \Vdash_{L,F} \Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$.*

Proof. If L is linear outside the filter, then Lemma 3 says that any frame satisfies axiom K. We then assume that L is not linear outside the filter. Thus, there are $a, b \in L \setminus F$ such that $a \not\leq b$ and $b \not\leq a$. Consider a frame $\mathcal{F} = \langle W, R\rangle$ with worlds w_1, w_2 and w_3 such that $w_1 R w_2$ and $w_1 R w_3$. Fix then a model M with frame $\mathcal{F}$ and such that $v_{w_2}(\varphi) = v_{w_3}(\varphi) = a$, $v_{w_2}(\psi) = b$ and $v_{w_3}(\psi) = a$. Now, observe that $v_{w_1}(\Box\psi) = a.b$ and $v_{w_1}(\Box\varphi) = a$. So we have $v_{w_1}(\Box\varphi \to \Box\psi) = a \supset a.b = a.b$ for $a \not\leq a.b$. Moreover $v_{w_1}(\Box(\varphi \to \psi)) = (a \supset b).(a \supset a) = b.1 = b$. Hence $v_{w_1}(\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)) = (b \supset a.b) = a.b \notin F$. □

The converse in Theorem 4 does not work without the assumption of strictly deductive. Possessing solely the implicative property, it is possible to identify a lattice L that is non-linear outside the filter, where the reasoning used in proving Theorem 4 is blocked. Consider the lattice L where there is a pair of values $a \notin F$ and $b \notin F$ such that, for some value f, $a \supset a = f \in F$, $f.b \leq a.b$. In this case, we have $v_{w_1}(\Box\varphi \to \Box\psi) = a \supset a.b = a.b$ and $v_{w_1}(\Box(\varphi \to \psi)) = (a \supset b).(a \supset a) = b.f$. This will give us $v_{w_1}(\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)) = (b.f \subset a.b) \in F$ since L is implicative. It is possible to verify that a lattice consisting solely of the elements necessary for this example possesses the K property.[6]

Let us now consider the case where implication $x \supset y$ is defined as the regular $-x + y$.

Lemma 4. *Let the lattice L be anti-monotone with down-distribution, M be a L-model and F be a set of designated values in L. If $x \supset y$ is defined as $-x + y$ and F is implicative, then every $w \in M$ satisfies ($\models_F$) the axiom K.*

Proof. Consider any w in M. Notably, $v_w(\Box\varphi) \leq v_{w'}(\varphi)$, for all wRw'. Thus, from anti-monotonicity, $-v_w(\Box\varphi) \geq -v_{w'}(\varphi)$, for all wRw'. Adding to both sides $v_{w'}(\psi)$, we obtain

$$-v_w(\Box\varphi) + v_{w'}(\psi) \geq v_{w'}(\varphi \to \psi)$$

This being valid for every w' accessed by w give us

$$\bigwedge\{-v_w(\Box\varphi) + v_{w'}(\psi) \mid wRw'\} \geq v_w(\Box(\varphi \to \psi))$$

Because L has down distribution,

$$-v_w(\Box\varphi) + \bigwedge\{v_{w'}(\psi) \mid wRw'\} \geq v_w(\Box(\varphi \to \psi))$$

$$v_w(\Box\varphi \to \Box\psi) \geq v_w(\Box(\varphi \to \psi))$$

As L is implicative, $w \Vdash \Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$. □

Theorem 5. *Let L be anti-monotone, involutive and with down-distribution, F be a set of designated values in L and $x \to y$ is defined as $-x + y$ in L. Then L is implicative if, and only if, every frame $\mathcal{F} = \langle W, R\rangle$ is such that $\mathcal{F} \Vdash_{L,F} \Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$.*

[6] Precisely, the example will be the lattice $L = \{0, a, b, f, 1\}$, $(a \supset b) = f$ and $x \supset y$ is strictly deductive for all cases where $x \neq a$ and $y \neq b$. The order in L is such that 1 is top value, 0 is bottom value and a, b and f are incomparable. All frames in this lattice with filter $\{f, 1\}$ satisfy axiom K. Examining this thoroughly is time-consuming and contributes little to the paper, so we leave it to the interested reader.

Proof. From Lemma 4, we obtain that every frame satisfies K if L is implicative. Let us then suppose that L is not implicative. Then there are $a,b \in L$ such that $a \leq b$ and $a \to b \notin F$.

Consider w accessing w_1 and w_2 such that $v_{w_1}(\varphi) = -a$, $v_{w_2}(\varphi) = -b$, $v_{w_1}(\psi) = a$, $v_{w_2}(\psi) = a$. Naturally, $v_w(\Box\psi) = a$ and, since L is anti-monotone, $v_w(\Box\varphi) = -b$. Hence $v_w(\Box\varphi \to \Box\psi) = --b + a = b$ because L is involutive. Moreover, we have $v_w(\Box(\varphi \to \psi)) = a$. Since $a \to b \notin F$, $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$ is not valid in w.

□

Without the anti-monotone or the down-distribution properties, we have too little control over the negation, and thus Lemma 4 will fail. But even with those properties, the converse of Theorem 5 fails without assuming that negation is involutive. The problem in this case is that we cannot use the double negation to extract the comparison we need like $v_w(\Box\varphi \to \Box\psi) = --b + a = b$.

5 An example

From a Boolean algebra $B = \langle D, +, ., - \rangle$ one may obtain the twist algebra T of B as suggested in [Vak77; Fid77]. The initial use for twist algebras was to provide semantics for Nelson logic [Nel49]. It has recently been used to model some paraconsistent set theories [CC21] and used for modal systems in [OR14]. The twist $T = \langle D \times D, +, ., - \rangle$ is defined with respect to B in such a way that for $a,b,c,d \in D$

1. $\langle a,b\rangle + \langle c,d\rangle = \langle a + c, b.d\rangle$.
2. $\langle a,b\rangle.\langle c,d\rangle = \langle a.c, b + d\rangle$.
3. $-\langle a,b\rangle = \langle b,a\rangle$.

The twist lattice has anti-monotone and involutive properties. In fact $--\langle a,b\rangle = -\langle b,a\rangle = \langle a,b\rangle$. Additionally, if $\langle a,b\rangle \leq \langle c,d\rangle$, then $a \leq c$ and $b \geq d$. So $-\langle a,b\rangle = \langle b,a\rangle \geq \langle d,c\rangle = -\langle c,d\rangle$. Note further that any restriction of T's domain will still have these properties so long as the restriction is closed by the operations $\{+, ., -\}$. Hence we have, for instance, anti-monotone and involutive properties for the restriction $P = \{\langle a,b\rangle \in D \times D \mid a + b = 1\}$; this is precisely the restriction used in [CC21] accounting for various paraconsistent systems. Notably, the restriction in P is such that a twist value $t + (-t)$ is always of the form $\langle 1,x\rangle$ while $t.(-t)$ is of the form $\langle x,1\rangle$. Assuming a filter $F = \{\langle 1,x\rangle \mid x \in D\}$ gives us a natural paraconsistent semantics. In this case, we should observe that the filter that gives us the axiom K is precisely the one Carnielli and Coniglio use in [CC21]:

Theorem 6. *Let T be the twist algebra obtained from the Boolean algebra $B = \langle D, +, ., - \rangle$ and $P = \{\langle a,b\rangle \in D \times D \mid a + b = 1\}$. Let F be a set of designated values in P, then $F \supset \{\langle x,y\rangle \mid x = 1\}$ if, and only if, every P-frame satisfies K.*

Proof. Because P is anti-monotone and involutive, Theorem 5 give us that if every frame satisfies K, then P is implicative in F. Note that, for any $x \in B$, $\langle 1,x\rangle \geq \langle 1,1\rangle$; so $(\langle 1,1\rangle \supset \langle 1,x\rangle) = \langle 1,x\rangle \in F$. On the other hand, suppose $F \supset \{\langle x,y\rangle \mid x = 1\}$. If $\langle a,b\rangle \leq \langle c,d\rangle$, then $a \leq d$, $b \geq c$ and $(\langle a,b\rangle \supset \langle c,d\rangle) = \langle c + b, a.d\rangle$. But, from P's definition, $a + b = 1$ and $c + d = 1$, thus we obtain $(\langle a,b\rangle \supset \langle c,d\rangle) = \langle 1,d\rangle \in F$. Hence P is implicative with filter F and every P-frame satisfies K. □

6 Final remarks

The topic addressed in this article is broad and we have focused on a limited segment of modal logics based on lattice semantics. We have studied some basic modal sentences that characterize algebraic properties of lattices. Indeed, in the presence of deductive implications, linearity is strictly connected to the validity of axiom K. Alternatively, for the regular implication defined semantically as $a \subset b = -a + b$, validity of K is connected to regularity properties for negation such as involutive and anti-monotone.

One notable advantage of restricting our analysis to lattice-based structures is that the underlying order relation offers a natural foundation for interpreting the necessity operator. As emphasized in [FM25], this becomes especially relevant when considering models in which different worlds operate according to distinct logics. A shared order relation allows for meaningful interaction across such worlds, facilitating cross-logical communication.

Finally, we have also expressed interest in investigating the conditions under which normal modal extensions exhibit the finite-model property. This feature is of particular significance for practical applications, such as the implementation of logical frameworks in computational systems.

Acknowledgments. M. Martins was partially supported by FCT – Fundação para a Ciência e a Tecnologia through project UIDB/04106/2025 at CIDMA.

References

[BJ19] Francesco Berto and Mark Jago. *Impossible worlds*. Oxford University Press, 2019.

[BMC14] Luís S. Barbosa, Manuel A. Martins, and Marta Carreteiro. "A Hilbert-style axiomatisation for equational hybrid logic". English. In: *J. Logic Lang. Inf.* 23.1 (2014), pp. 31–52. DOI: 10.1007/s10849-013-9184-6.

[CC21] Walter A. Carnielli and Marcelo E. Coniglio. "Twist-Valued Models for Three-Valued Paraconsistent Set Theory". In: *Logic and Logical Philosophy* 30.2 (2021), 187–226. DOI: 10.12775/LLP.2020.015.

[DS07] Răzvan Diaconescu and Petros Stefaneas. "Ultraproducts and possible worlds semantics in institutions". English. In: *Theor. Comput. Sci.* 379.1-2 (2007), pp. 210–230. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2007.02.068.

[Fid77] Manuel Fidel. "An algebraic study of a propositional system of Nelson". In: *Proceedings of the First Brazilian Conference on Mathematical Logic*. Marcel Dekker inc., 1977.

[Fit91] Melvin C. Fitting. "Many-valued modal logics". In: *Fundamenta informaticae* 15.3-4 (1991), pp. 235–254.

[FM25] Alfredo R. Freire and Manuel A. Martins. "Modality across different logics". In: *Logic Journal of the IGPL* 33.3 (2025), jzae082. ISSN: 1367-0751. DOI: 10.1093/jigpal/jzae082.

[FMM24] Alfredo R. Freire, Manuel A. Martins, and Alexandre Madeira. "The Institution of Many-Logics Modal Logic". In: *Recent Trends in Algebraic Development Techniques - 27th IFIP WG 1.3 International Workshop, WADT 2024, Enschede, The Netherlands, July 8, 2024, Revised Selected Papers*. Ed. by Ionuț Țuțu. Vol. 15587. Lecture Notes in Computer Science. Springer, 2024, pp. 94–110. DOI: 10.1007/978-3-031-88930-1_5.

[KJ21] Alexander W. Kocurek and Ethan J. Jerzak. “Counterlogicals as Counterconventionals”. In: *Journal of Philosophical Logic* 50.4 (2021), pp. 673–704. DOI: 10.1007/S10992-020-09581-6.

[Kri59] Saul Kripke. “A Completeness Theorem in Modal Logic”. In: *Journal of Symbolic Logic* 24.1 (1959), pp. 1–14. DOI: 10.2307/2964568.

[MMH19] Maria Manzano, Manuel A. Martins, and Antonia Huertas. “Completeness in equational hybrid propositional type theory”. In: *Studia Logica* 107.6 (2019), pp. 1159–1198. ISSN: 0039-3215,1572-8730. DOI: 10.1007/s11225-018-9833-5.

[Nel49] David Nelson. “Constructible falsity”. In: *The Journal of Symbolic Logic* 14.1 (1949), pp. 16–26. DOI: 10.2307/2266838.

[Nol13] Daniel P. Nolan. “Impossible Worlds”. In: *Philosophy Compass* 8.4 (2013), pp. 360–372. DOI: https://doi.org/10.1111/phc3.12027.

[OL12] Sergei P. Odintsov and E.I. Latkin. “BK-lattices. Algebraic semantics for Belnapian modal logics”. In: *Studia Logica* 100.1-2 (2012), pp. 319–338. DOI: 10.1007/S11225-012-9380-4.

[OR14] Hiroakira Ono and Umberto Rivieccio. “Modal twist-structures over residuated lattices”. In: *Logic Journal of IGPL* 22.3 (2014), pp. 440–457. DOI: 10.1093/jigpal/jzt043.

[Pri08a] Graham Priest. *An introduction to non-classical logic: From if to is*. Cambridge University Press, 2008.

[Pri08b] Graham Priest. “Many-valued modal logics: a simple approach”. In: *The Review of Symbolic Logic* 1.2 (2008), 190–203. DOI: 10.1017/S1755020308080179.

[Vak77] Dimiter Vakarelov. “Notes on N-lattices and constructive logic with strong negation”. In: *Studia Logica* 36.1/2 (1977), pp. 109–125. DOI: 10.1007/BF02121118.

Monoid Structures on Indexed Containers

Michele De Pascalis
Tallinn University of Technology, Estonia
michde@taltech.ee

Tarmo Uustalu
Reykjavik University, Iceland
Tallinn University of Technology, Estonia
tarmo@ru.is

Niccolò Veltrì
Tallinn University of Technology, Estonia
niccolo@cs.ioc.ee

Containers represent a wide class of type constructions that are relevant for functional programming and (co)inductive reasoning. Indexed containers generalize this notion to better fit the scope of dependently typed programming. When interpreting types to be sets, a container describes an endofunctor on the category of sets while an I-indexed container describes an endofunctor on the category Set^I of I-indexed families of sets.

We consider the monoidal structure on the category of I-indexed containers whose tensor product of containers describes the composition of the respective induced endofunctors. We then give a combinatorial characterization of monoids in this monoidal category, and we show how these monoids correspond precisely to monads on the induced endofunctors on Set^I. Lastly, we conclude by presenting some examples of monads on Set^I that fall under our characterization, including the product of two monads, indexed variants of the state and the writer monads and an example of a free monad. The technical results of this work are accompanied by a formalization in the proof assistant Cubical Agda.

1 Introduction

Abbott et al. [1] introduced the concept of (simple, non-indexed) containers as a representation for a wide class of parameterized datatypes, corresponding to strictly positive types, a class of set endofunctors. The subsequent indexed generalization of containers by Altenkirch et al. [6] covers a class of functors $\mathsf{Set}^I \to \mathsf{Set}^J$ between families of sets for I, J some given sets of indices.

An example of datatype arising from an indexed container (in which the index sets I and J are both equal to the set of natural numbers Nat) is the type of vectors with bounded size. In proof assistants such as Agda, the type $\mathsf{Vec}\,A\,n$ of vectors with length $\leq n$ can be implemented as an inductive type:

$$\begin{aligned}&\mathsf{data}\ \mathsf{Vec}\ (A : \mathsf{Set}) : \mathsf{Nat} \to \mathsf{Set}\ \mathsf{where}\\ &\quad \mathsf{nil} : \{n : \mathsf{Nat}\} \to \mathsf{Vec}\,A\,n\\ &\quad \mathsf{cons} : \{n : \mathsf{Nat}\} \to A \to \mathsf{Vec}\,A\,n \to \mathsf{Vec}\,A\,(1+n)\end{aligned}$$

But this can be equivalently defined as the type $\sum_{k\leq n} \mathsf{Fin}\ k \to A$, whose elements are pairs of a number k smaller than n (the actual length of the vector) and a function from $\mathsf{Fin}\ k$ (the type of numbers $< k$) to A, i.e. a k-tuple of elements of A. This type arises from a Nat-indexed container with shapes at index n being the numbers $k \leq n$ and positions in shape k being elements of $\mathsf{Fin}\ k$.

Indexed containers are essentially a notational variant of the (generalized) polynomials of Gambino and Hyland [12], a concept definable for any locally Cartesian closed category $\mathscr{C}$. A polynomial from I to

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. 37–54, doi:10.4204/EPTCS.430.4

J interprets into a generalized polynomial functor $\mathscr{C}/I \to \mathscr{C}/J$ between slice categories, where $\mathscr{C} := \mathsf{Set}$ is the archetypical choice of $\mathscr{C}$. Polynomials in this sense are a variation on Joyal's [16] species, which interpret into analytic functors. Weber [26] showed that the locally Cartesian closed category can be replaced by a category with pullbacks.

The container and polynomial view of the same concept complement each other. The polynomial view has advantages for theoretical developments, in particular generalizations. For working with concrete examples or formalizing the theory and examples in a type-theoretical proof assistant, the container view tends to be smoother as argued e.g. by Finster et al. [11].

Monads as endofunctors with a monoid structure are ubiquitous in both type theory and category theory, with the type-theoretical (dependently typed programming) uses most often coming from programming language theory. Polynomial functors with a monad structure, called polynomial monads, are an important special case and their in-depth study was pioneered by Gambino and Kock [13].

Works on polynomials and containers also show variety in what they choose to be the appropriate notion of polynomial/container morphism. Abbott et al. [1], Altenkirch et al. [6] and Ahman et al. [3, 4] consider general natural transformations between the corresponding functors, while *e.g.* Finster et al. [11] only target Cartesian strong natural transformations, as noted in their Section 2.2. Gambino and Kock [13] describe general natural transformations when defining polynomials, but only consider Cartesian monads, *i.e.* monads whose unit and multiplication are Cartesian natural transformations.

Cartesian polynomial monads are of particular interest in the meta-theory of type theory because they arise in the semantics of type theory. They help organize Awodey's [7] natural models of type theory, as demonstrated by Awodey and Newstead [8, 21] and Aberlé and Spivak [2].

Being a monad is not a mere property, there can be several choices of unit and multiplication that make an endofunctor into a monad. Hence it is natural to wonder whether, for container monads, these choices can be beneficially analyzed in terms of their container representation. Uustalu [24] answered this question for non-indexed containers by unpacking what a monoid structure on a container is by definition and transforming the result to a somewhat simpler form.

Our contribution The contribution of this work is to extend the work of Uustalu [24] to indexed containers and to provide examples of this characterization.

In detail, we outline the monoidal structure on indexed containers that corresponds to composition of their extents. We then generalize Uustalu's characterization to account for indexing by appropriately adjusting the data and equations and prove that the resulting structures correspond precisely to monad structures on indexed container functors.

Lastly, we show how this characterization can be used practically, by employing it to describe the product of two indexed container monads, as well as indexed variants of the well-known state monad and writer monads [5] and, as an example of a free monad, the free monad which, when applied to the family Fin, gives the family of well-scoped λ-terms.

We formalize this development in Cubical Agda.

Related work Gambino and Kock [13] studied the bicategorical structure of (generalized) polynomials in arbitrary locally Cartesian closed categories: given an LCCC $\mathscr{C}$ and objects I, J of $\mathscr{C}$, their category $\mathsf{Poly}_{\mathscr{C}}(I,J)$ embeds fully and faithfully into the functor category $[\mathscr{C}/I, \mathscr{C}/J]_{\text{strong}}$ where the strengths are wrt. the canonical actions of $\mathscr{C}$ with its product monoidal structure on $\mathscr{C}/I$ and $\mathscr{C}/J$. Furthermore, one can define a bicategory whose objects are objects in $\mathscr{C}$ and whose hom-categories are given by $\mathsf{hom}(I,J) := \mathsf{Poly}_{\mathscr{C}}(I,J)$. Then, the previously described family of embeddings comes together into a

locally fully faithful 2-functor into the sub-2-category of Cat having $\mathscr{C}$-slices as 0-cells, polynomial functors as 1-cells, and strong natural transformations as 2-cells.

In our work, we consider indexed containers where both indexings are over the same fixed set I. This restriction corresponds in [13] to considering the full sub-bicategory of $\mathsf{Poly}_{\mathscr{C}}$ whose only 0-cell is I, i.e., a monoidal category which is analogous to the one studied in this paper. It is also worth noting that Gambino and Kock considered polynomial monads whose unit and multiplication are Cartesian, while our characterization is of general monads on extents of containers.

Proof assistant formalizations, mostly in Agda, of different chapters of the theory of containers and polynomials have been carried out by various groups of researchers. Ahman et al. [3] formalized directed containers (comonoids in containers). Finster et al. [11] formalized dependent polynomials/indexed containers. Damato et al. [9] formalized initial algebra and final coalgebra constructions in containers. Joram and Veltri [15] developed a version of containers with symmetries, called action containers. Purdy and Damato [22] have recently come up with a combinatorial characterization of distributive laws between monads arising from (non-indexed) containers.

Structure of the paper In Section 2 we review the category of indexed containers, that we call IC_I (with I being the indexing set), as well as its composition monoidal structure. In Section 3 we recollect the notion of monoid internal to a monoidal category and provide a combinatorial characterization of monoids in IC_I. Section 4 collects a number of examples of interest in dependently-typed functional programming, all introduced using our combinatorial approach: closure under Cartesian products, indexed variants of the state and the writer monads, and untyped λ-terms as an instance of a particular free monad. We conclude in Section 5, where we discuss some ideas for future work.

Formalization This work has been formalized in the Cubical Agda proof assistant. The choice of Cubical Agda is motivated by its support for functional extensionality, which is employed in our development, crucially in the proof of fully-faithfulness of the interpretation in endofunctors and in proving the combinatorial monoid equations for the examples in Section 4. Moreover, the use of Cubical Agda readily gives us the possibility of extending our work from sets to groupoids and higher-dimensional types.
The code is freely available at `https://github.com/mikidep/indexed-monads/tree/lsfa2025`, or as clickable HTML at `https://mikidep.github.io/indexed-monads/lsfa25/Everything.html`. For the reader's convenience, the formalized proofs and examples are decorated with a direct hyperlink, in the form of the Agda logo ().

Notational conventions We write composition of functions, and more generally morphisms in categories, in diagrammatic order: the composition of morphisms $A \xrightarrow{f} B \xrightarrow{g} C$ is denoted by $f\,;g$. We use curly braces for implicit arguments of dependent functions. When we want to make an implicit argument visible, we add it as a subscript. For example, given $f : \prod_{\{x:X\}} \prod_{y:Y} Z\ x\ y$, we denote its application to the implicit argument $x : X$ and the explicit argument $y : Y$ by $f_x\ y : Z\ x\ y$. An underscore appearing in the left-hand side of a function definition represents an argument that is not used in the definition, and therefore does not appear in the right-hand side. Set denotes the usual category of sets and functions. In Cubical Agda, a type X is a set if, for all $x, y : X$, the equality type $x \equiv y$ is a proposition, i.e. for all $p, q : x \equiv y$, we have $p \equiv q$. Given a set I, Set^I denotes the category of presheaves over I considered as a discrete category. The objects of this category are families of sets, i.e., functions of type $I \to \mathsf{Set}$. The morphisms from X to Y are functions of type $\prod_{\{i:I\}} X\ i \to Y\ i$. We write $X \to^I Y$ for the homset

$\mathsf{Set}^I(X,Y)$. Identity in Set^I is denoted id^{I} while composition is $;^I$. That is, we define $(f\,;^I\,g)_i := f_i\,;\,g_i$ for $f : X \to^I Y$, $g : Y \to^I Z$. The singleton set is denoted $\mathbb{1}$ with $\star$ as unique inhabitant.

2 Indexed containers

We are interested in endofunctors on Set^I which are describable as the extents of *indexed containers* in the sense of Altenkirch et al. [6]. Note that in general the notion of indexed container is parameterized in two indexing sets, however, in this work we only deal with polynomials describing endofunctors, for which the two parameters have to coincide. We shall recap the relevant definitions here.

Definition 2.1 (Indexed container, cf. ICont^* in [6] 🙄)**.** An *indexed container* (denoted $S \lhd P$) is composed of an indexed set $S : I \to \mathsf{Set}$, together with a family of indexed sets over it, $P : \prod_{\{i:I\}} S\,i \to I \to \mathsf{Set}$.

Definition 2.2 (Extent of an indexed container 🙄)**.** Let $S \lhd P$ be an indexed container. We define its *extent* $[\![S \lhd P]\!]$ as the endofunctor on Set^I defined on objects by

$$[\![S \lhd P]\!]\,X\,i := \sum_{s:S\,i} P_i\,s \;\to^I X$$

and on maps by

$$[\![S \lhd P]\!] : (X \to^I Y) \to ([\![S \lhd P]\!]\,X \to^I [\![S \lhd P]\!]\,Y)$$
$$([\![S \lhd P]\!]\,f)_i\,(s,v) := (s, v\,;^I\,f)$$

Definition 2.3 (Morphism of indexed containers 🙄)**.** Let $S \lhd P$ and $S' \lhd P'$ be indexed containers. A morphism between them is a function of type $\prod_{\{i:I\}} \prod_{s:S\,i} [\![S' \lhd P']\!](P_i\,s)\,i$.

The type of container morphisms $\mathsf{IC}_I(S \lhd P, S' \lhd P')$ can be unfolded as follows:

$$\begin{aligned}
&\prod_{\{i:I\}} \prod_{s:S\,i} [\![S' \lhd P']\!](P_i\,s)\,i \\
&= \prod_{\{i:I\}} \prod_{s:S\,i} \sum_{s':S'\,i} P'_i\,s' \to^I P_i\,s \\
&\simeq \sum_{\sigma:S \to^I S'} \prod_{\{i:I\}} \prod_{s:S\,i} P'_i\,(\sigma\,i\,s) \to^I P_i\,s
\end{aligned}$$

Given $f : \mathsf{IC}_I(S \lhd P, S' \lhd P')$, we denote as $\sigma\,f$ and $\pi\,f$ the two components of the image of f under the last equivalence above.

Indexed containers and their morphisms form a category IC_I with identity and composition defined as follows:

$$\begin{aligned}
&(\mathsf{id}_{S \lhd P})_i\,s := (s, \mathsf{id}_{P_i s}) \\
&(\alpha\,;\beta)_i\,s := \mathsf{let} \\
&\qquad (s', v') \leftarrow \alpha_i\,s \\
&\qquad (s'', v'') \leftarrow \beta_i\,s' \\
&\quad \mathsf{in}\,(s'', v''\,;^I\,v')
\end{aligned}$$

The extent operation $[\![-]\!]$ extends to a functor from IC_I to $\mathsf{Endo}(\mathsf{Set}^I)$, the category with objects given by endofunctors on Set^I and morphisms by natural transformations. The following lemma was already proved in [6] for non-indexed containers.

Lemma 2.1. $[\![-]\!]$ *is full and faithful.*

Proof . The proof employs (co)end calculus. For a reference, see [19]. Also, note that ends over discrete categories coincide with products. A more direct proof is carried out in the formalization.

$$
\begin{aligned}
&\mathsf{Nat}\left([\![S \triangleleft P]\!], [\![S' \triangleleft P']\!]\right) \\
\simeq &\int_X \mathsf{Set}^I\left([\![S \triangleleft P]\!]\, X, [\![S' \triangleleft P']\!]\, X\right) && \text{(nat. transfs. as an end)} \\
\stackrel{\mathsf{def}}{=} &\int_X \prod_{i:I} \mathsf{Set}\left([\![S \triangleleft P]\!]\, X\ i, [\![S' \triangleleft P']\!]\, X\ i\right) \\
\stackrel{\mathsf{def}}{=} &\int_X \prod_{i:I} \mathsf{Set}\left(\sum_{s:S\ i} \mathsf{Set}^I(P\ s, X), [\![S' \triangleleft P']\!]\, X\ i\right) \\
\simeq &\int_X \prod_{i:I} \prod_{s:S\ i} \mathsf{Set}\left(\mathsf{Set}^I(P\ s, X), [\![S' \triangleleft P']\!]\, X\ i\right) && \text{(dependent (un)currying)} \\
\simeq &\prod_{i:I} \prod_{s:S\ i} \int_X \mathsf{Set}\left(\mathsf{Set}^I(P\ s, X), [\![S' \triangleleft P']\!]\, X\ i\right) && \text{(Fubini rule for ends)} \\
\simeq &\prod_{i:I} \prod_{s:S\ i} \mathsf{Nat}\left(\mathsf{Set}^I(P\ s, -), [\![S' \triangleleft P']\!]\, (-)\ i\right) && \text{(nat. transfs. as an end)} \\
\simeq &\prod_{i:I} \prod_{s:S\ i} [\![S' \triangleleft P']\!](P\ s)\ i && \text{(Yoneda)}
\end{aligned}
$$

□

2.1 Monoidal categories and strong monoidal functors

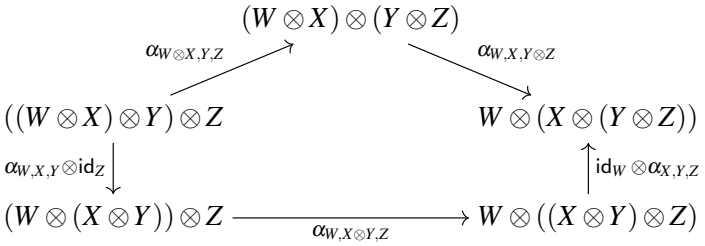
Monoidal categories [20, 17] are ubiquitous in category theory at large, and in particular in its applications in computer science. We quickly recap the notions we employ, and in doing so specify the notation and conventions we adopt in the rest of the work.

A *monoidal category* $(\mathscr{C}, \mathsf{I}, \otimes)$ consists of a category $\mathscr{C}$, together with an object $\mathsf{I} : \mathscr{C}$, a bifunctor $\otimes : \mathscr{C} \times \mathscr{C} \to \mathscr{C}$ and natural isomorphisms typed

$$
\begin{aligned}
\lambda_X &: \mathsf{I} \otimes X \xrightarrow{\sim} X \\
\rho_X &: X \otimes \mathsf{I} \xrightarrow{\sim} X \\
\alpha_{X,Y,Z} &: (X \otimes Y) \otimes Z \xrightarrow{\sim} X \otimes (Y \otimes Z)
\end{aligned}
$$

For these, the following two diagrams have to commute for any choice of X, Y, Z, W:

$$
\begin{array}{ccc}
(X \otimes \mathsf{I}) \otimes Y & \xrightarrow{\quad\alpha_{X,\mathsf{I},Y}\quad} & X \otimes (\mathsf{I} \otimes Y) \\
& {\scriptstyle \rho_X \otimes \mathsf{id}_Y} \searrow \quad \swarrow {\scriptstyle \mathsf{id}_X \otimes \lambda_Y} & \\
& X \otimes Y &
\end{array}
\qquad \text{(triangle eq.)}
$$



(pentagon eq.)

Given two monoidal categories $(\mathscr{C}, \mathsf{I}, \otimes)$ and $(\mathscr{D}, \mathsf{I}', \otimes')$, a *(strong) monoidal functor* consists of a functor $F : \mathscr{C} \to \mathscr{D}$, equipped with

- an isomorphism $\varepsilon : \mathsf{I}' \xrightarrow{\sim} F\ \mathsf{I}$,
- a natural isomorphism $\psi_{X,Y} : F\ X \otimes' F\ Y \xrightarrow{\sim} F\ (X \otimes Y)$.

2.2 The composition monoidal structure of indexed containers

IC_I features a composition monoidal category structure, which we shall describe here explicitly.

Definition 2.4 (Identity container 🙃)**.** The *identity container* I is defined as $S \triangleleft P$, where:

$$S\ i := \mathbb{1}$$
$$P_i \star j := i \equiv j$$

Definition 2.5 (Composite container 🙃)**.** Let $S^0 \triangleleft P^0$ and $S^1 \triangleleft P^1$ be indexed containers over I. Their composition $(S^0 \triangleleft P^0) \otimes (S^1 \triangleleft P^1)$ is defined as $S \triangleleft P$ where

$$S\ i := [\![S^0 \triangleleft P^0]\!]\ S^1\ i$$
$$P_i\ (s, s')\ k := \sum_{j:I} \sum_{p' : P^0_i\ s\ j} P^1_j\ (s'\ p')\ k$$

Lemma 2.2. $(\mathsf{IC}_I, \mathsf{I}, \otimes)$ *is a monoidal category.*

Proof 🙃. Functoriality of $\otimes$ holds definitionally, while the unitors and associator (and their inverses) are defined by reassociating Σ-types and providing the necessary refl's when needed. All is carried out in detail in the formalization. □

Lemma 2.3. $[\![-]\!]$ *is a strong monoidal functor* $(\mathsf{IC}_I, \mathsf{I}, \otimes) \to (\mathsf{Endo}(\mathsf{Set}^I), \mathsf{Id}, (\circ))$.

Proof 🙃. Similarly to the above, this is all reassociations and identity type bookkeeping, please refer to the formalization. □

3 Monoids, indexed containers and monads

3.1 Monoid objects

We recall here the definition of a monoid object in a monoidal category.

Definition 3.1 (Monoid in a monoidal category)**.** Let $(\mathscr{C}, \mathsf{I}, \otimes)$ be a monoidal category. A *monoid* (X, η, μ) in $\mathscr{C}$ is an object X of $\mathscr{C}$ coming with two maps $\eta : \mathsf{I} \to X$ and $\mu : X \otimes X \to X$, making the following diagrams commute.

$$\begin{array}{ccccc} \mathsf{I} \otimes X & \xrightarrow{\eta \otimes \mathsf{id}} & X \otimes X & \xleftarrow{\mathsf{id} \otimes \eta} & X \otimes \mathsf{I} \\ & \searrow^{\lambda_X} & \downarrow{\scriptstyle \mu} & \swarrow_{\rho_X} & \\ & & X & & \end{array} \qquad \text{(unitality)}$$

$$\begin{array}{ccc} (X \otimes X) \otimes X & \xrightarrow{\alpha_{X,X,X}} & X \otimes (X \otimes X) \\ \downarrow{\scriptstyle \mu \otimes \mathsf{id}} & & \downarrow{\scriptstyle \mathsf{id} \otimes \mu} \\ X \otimes X & & X \otimes X \\ & \searrow^{\mu} \quad \swarrow_{\mu} & \\ & X & \end{array} \qquad \text{(associativity)}$$

Definition 3.2 (Monoid morphisms)**.** Let (X, η, μ) and (Y, η', μ') be monoids in $\mathscr{C}$. A *monoid morphism* between them is a morphism $f : \mathscr{C}(X, Y)$, making the following diagrams commute:

$$\begin{array}{ccc} \mathsf{I} & \xrightarrow{\eta} & X \\ & \searrow^{\eta'} & \downarrow{\scriptstyle f} \\ & & Y \end{array} \qquad \begin{array}{ccc} X \otimes X & \xrightarrow{\mu} & X \\ \downarrow{\scriptstyle f \otimes f} & & \downarrow{\scriptstyle f} \\ Y \otimes Y & \xrightarrow[\mu']{} & Y \end{array} \qquad (1)$$

It can be shown that identities and compositions of monoidal morphisms are themselves monoidal morphisms. Hence, monoids in a monoidal category $(\mathscr{C}, \mathsf{I}, \otimes)$ together with the above defined morphisms form in turn a category, denoted $\mathsf{Mon}(\mathscr{C})$. The following lemma is relevant to the main result of this work.

Lemma 3.1. *Let $(\mathscr{C}, I, \otimes)$ and $(\mathscr{D}, \mathsf{I}', \otimes')$ be monoidal categories, and let $F : \mathscr{C} \to \mathscr{D}$ with $\varepsilon : I' \to FI$ and $\psi_{X,Y} : FX \otimes' FY \to F(X \otimes Y)$ be a full, faithful and strong monoidal functor. Then, F induces a full and faithful functor between* $\mathsf{Mon}(\mathscr{C})$ *and* $\mathsf{Mon}(\mathscr{D})$*,* i.e. $\mathsf{Mon}(\mathscr{C})$ *is a full subcategory of* $\mathsf{Mon}(\mathscr{D})$*.*

Proof. Let (X, η, μ) be a monoid. It is well-known that F, being lax monoidal, induces a monoid structure on FX with:

$$\eta' := \mathsf{I}' \xrightarrow{\varepsilon} F\mathsf{I} \xrightarrow{F\eta} FX$$
$$\mu' := FX \otimes' FX \xrightarrow{\psi_{X,X}} F(X \otimes X) \xrightarrow{F\mu} FX$$

F being strong monoidal, ε and ψ are natural isomorphisms, therefore their precompositions $(\varepsilon\,;-)$ and $(\psi\,;-)$ are in turn isomorphisms, so we have:

$$\begin{array}{ll} \mathscr{D}(FX \otimes' FX, FX) & \\ \simeq \mathscr{D}(F(X \otimes X), FX) & (\psi_{X,X}\,;-) \text{ is an iso} \\ \simeq \mathscr{C}(X \otimes X, X) & F \text{ is full and faithful} \end{array}$$

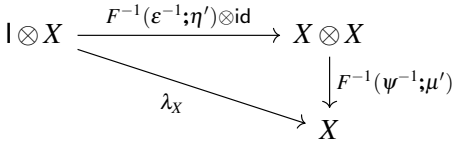
Notice that the right-to-left function underlying this isomorphism constructs the multiplication operation $\mu' : \mathscr{D}(FX \otimes' FX, FX)$ described above from a multiplication operation $\mu : \mathscr{C}(X \otimes X, X)$.

Analogously, for ε we have $\mathscr{D}(\mathsf{I}', FX) \simeq \mathscr{C}(\mathsf{I}, X)$.

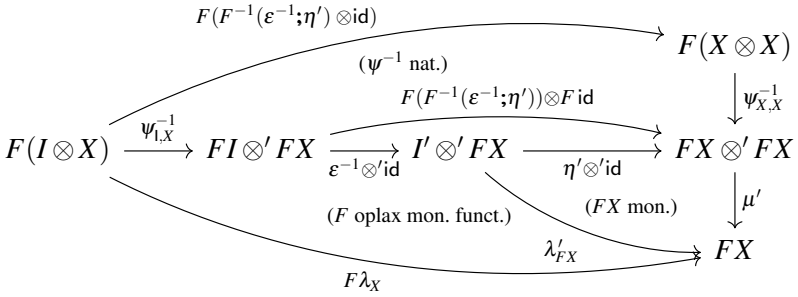
It is only left to prove that under the inverses the monoid equations are still satisfied. Let (FX, η', μ') be a monoid in $\mathscr{D}$. Walking through the above isomorphism chain, this gets mapped to:

$$(X, F^{-1}(\varepsilon^{-1}\,;\eta'), F^{-1}(\psi^{-1}_{X,X}\,;\mu'))$$

This has to satisfy the monoid equations from Definition 3.1 in $\mathscr{C}$. We will only report here one of the unitality proofs, the rest of them can be worked out similarly. The following diagram has to commute.



Since F is faithful, it suffices to show that the image of the diagram under F commutes. That is achieved by the following pasting:



Lastly, it can be shown that for fixed $M, M' : \mathsf{Mon}(\mathscr{C})$, the (bijective) action of F on $\mathsf{Mon}(\mathscr{C})(M, M')$ respects and reflects the commutative diagrams in (1). □

3.2 Monoid structures on indexed containers

In the specific monoidal category of indexed containers described in Subsection 2.2, monoids can be presented in a more combinatorial fashion.

Definition 3.3 (ICMS 🦔)**.** Let $S \triangleleft P$ be an indexed container. An *indexed container monoid structure (*ICMS*)* on $S \triangleleft P$ consists of the following data:

$$\begin{aligned}
\mathsf{e} &: \prod_{\{i:I\}} S\ i \\
\bullet &: \prod_{\{i:I\}} \prod_{s:S\ i} (P_i\ s \to^I S) \to S\ i \\
P\mathsf{e}_{\equiv} &: \prod_{\{i:I\}} \prod_{\{j:I\}} P_i\ \mathsf{e}_i\ j \to i \equiv j \\
\uparrow &: \prod_{\{i:I\}} \prod_{\{s:Si\}} \prod_{\{s':P_i\,s\to^I S\}} \prod_{\{j:I\}} P_i\ (s \bullet s')\ j \to I \\
\nwarrow &: \prod_{\{i:I\}} \prod_{\{s:Si\}} \prod_{\{s':P_i\,s\to^I S\}} \prod_{\{j:I\}} \prod_{p:P_i\,(s\bullet s')\,j} P_i\ s\ (\uparrow p) \\
\nearrow &: \prod_{\{i:I\}} \prod_{\{s:Si\}} \prod_{\{s':P_i\,s\to^I S\}} \prod_{\{j:I\}} \prod_{p:P_i\,(s\bullet s')\,j} P_{\uparrow p}\ (s'_{\uparrow p}\ (\nwarrow p))\ j
\end{aligned}$$

Arguments i, s, s', j for $\uparrow$, $\nwarrow$ and $\nearrow$ are implicit as they can be often inferred by the type of p. However, on occasion, we might specify them as subscripts, in their respective order.

For any index $i : I$, shape $s : S\ i$ and index $j : I$, we require

$$s \bullet_i \mathsf{e}^P \equiv s \qquad \text{(e-unit-l)}$$

$$\nwarrow_{i,s,\mathsf{e}^P,j}\ p \equiv p \qquad \forall p : P_i\ (s \bullet \mathsf{e}^P)\ j \qquad (\nwarrow\text{-unit-l})$$

$$\mathsf{e}_i \bullet_i \overline{s} \equiv s \qquad \text{(e-unit-r)}$$

$$\nearrow_{i,\mathsf{e}_i,\overline{s},j}\ p \equiv p \qquad \forall p : P_i\ (\mathsf{e}_i \bullet \overline{s})\ j \qquad (\nearrow\text{-unit-r})$$

where

$$\begin{aligned} \mathsf{e}^P &: P_i\ s \to^I S \\ \mathsf{e}^P_j\ \_ &:= \mathsf{e}_j \\ \overline{s} &: P_i\ \mathsf{e}_i \to^I S \\ \overline{s}_j\ \_ &:= s \end{aligned}$$

For any index $i : I$, families of shapes $s : S\ i, s' : P_i\ s \to^I S, s'' : \prod_{\{k:I\}} \prod_{q:P_i\ s\ k} P_k\ (s'_k\ q) \to^I S$, index $j : I$ and position $p : P_i((s \bullet s') \bullet \overline{s''})\ j$, we require

$$(s \bullet_i s') \bullet_i \overline{s''} \equiv s \bullet_i (s' \bullet^P s'') \qquad (\bullet\text{-assoc})$$

$$\uparrow_{i,s\bullet_i s',\overline{s''},j}\ p \equiv \uparrow_{\uparrow p,s'(\nwarrow p),s''(\nwarrow p),j}\ (\nearrow_{i,s,s'\bullet^P s'',j}\ p) \qquad (\uparrow\text{-}\nearrow\uparrow\text{-assoc})$$

$$\uparrow_{i,s,s',\uparrow p}\ (\nwarrow_{i,s\bullet_i s',\overline{s''},j}\ p) \equiv \uparrow_{i,s,s'\bullet^P s'',j}\ p \qquad (\nwarrow\uparrow\text{-}\uparrow\text{-assoc})$$

$$\nwarrow_{i,s,s',\uparrow p}\ (\nwarrow_{i,s\bullet_i s',\overline{s''},j}\ p) \equiv \nwarrow_{i,s,s'\bullet^P s'',j}\ p \qquad (\nwarrow\nwarrow\text{-}\nwarrow\text{-assoc})$$

$$\nearrow_{i,s,s',\uparrow p}\ (\nwarrow_{i,s\bullet_i s',\overline{s''},j}\ p) \equiv \nwarrow_{\uparrow p,s'(\nwarrow p),s''(\nwarrow p),j}\ (\nearrow_{i,s,s'\bullet^P s'',j}\ p) \qquad (\nwarrow\nearrow\text{-}\nearrow\nwarrow\text{-assoc})$$

$$\nearrow_{i,s\bullet_i s',\overline{s''},j}\ p \equiv \nearrow_{\uparrow p,s'(\nwarrow p),s''(\nwarrow p),j}\ (\nearrow_{i,s,s'\bullet^P s'',j}\ p) \qquad (\nearrow\text{-}\nearrow\nearrow\text{-assoc})$$

where

$$\begin{aligned} s' \bullet^P s'' &: P_i\ s \to^I S \\ (s' \bullet^P s'')_k\ q &:= s'_k\ q \bullet_k s''_k\ q \\ \overline{s''} &: P_i\ (s \bullet_i s') \to^I S \\ \overline{s''}_\ell\ r &:= (s''_{\uparrow r}\ (\nwarrow r))_\ell\ (\nearrow r) \end{aligned}$$

We will refer to the type of such structures as $\mathsf{ICMS}(S \triangleleft P)$.

Comparing this with the characterization of containers with monad structures in Uustalu's work [24, Section 3], it is evident that every piece of data and every equation outlined there reappears in Definition 3.3, adapted to carry the necessary indices. However, the following pieces of data are new to the indexed case:

- $P\mathsf{e}_{\equiv i,j}\ p$ is an extra coherence required of positions p on all shapes e_i, arising from unitality: their source index must be the same as their target index.
- $\uparrow p$ fulfills a similar function as $\nwarrow p$: where the latter maps a position in the composite shape to a position in the outer shape, the former specifies this new position's index.

- (↑-↗↑-assoc) and (↖↑-↑-assoc) are extra coherences deriving from how associativity acts on indices.

It is also worth noting that, as it was in [24], many of the equations are heterogeneous, *i.e.* the types of the equated terms only match because of other non-definitional equations. As an instance, in equation (↖↖-↖-assoc), the left-hand side has type $P\ s\ (\uparrow(\nwarrow p))$, while the right-hand side has type $P\ s\ (\uparrow p)$: the two types are equal since $\uparrow(\nwarrow p)$ and $\uparrow p$ are equal by (↖↑-↑-assoc). Additionally, the same terms often are required to have propositionally, but not definitionally equal types, in the equated expressions, *e.g.* in equation (↖↗-↗↖-assoc), p should simultaneously have types $P\ ((s \bullet s') \bullet \overline{s''})\ j$ and $P\ (s \bullet (s' \bullet^P s''))\ j$, which are provably equal due to (•-assoc).

Such heterogeneous equalities can also be found in the non-indexed case [24]. However, some new ones appear due to $P\mathsf{e}_{\equiv}$, which are less trivial. In particular, for any $i : I, s : S\ i$ and $j : J$ we have

$$P\mathsf{e}_{\equiv i,j}\ (\nearrow p) : \uparrow p \equiv j \qquad \forall p : P_i\ (s \bullet \mathsf{e}^P)\ j$$
$$P\mathsf{e}_{\equiv i,j}\ (\nwarrow p) : i \equiv \uparrow p \qquad \forall p : P_i\ (\mathsf{e}_i \bullet \overline{s})\ j$$

The equations (↖-unit-l) and (↗-unit-r) type-check thanks to these two derivable equality proofs.

As should be clear by now, Definition 3.3 is constructed *ad-hoc* to capture the combinatorial structure of monoids in $(\mathsf{IC}_I, \mathsf{I}, \otimes)$. In fact, the following holds:

Lemma 3.2 (Characterization of monoids)**.** *Let $S \triangleleft P$ be an indexed container. The type of monoids structures in $(\mathsf{IC}_I, \otimes, \mathsf{I})$ on $S \triangleleft P$ is equivalent to $\mathsf{ICMS}(S \triangleleft P)$.*

Proof 🕮. We will illustrate how the data of an ICMS is translated into the corresponding monoid structure. To that end, assume we have an $\mathsf{ICMS}(S \triangleleft P)$. We shall use it to define two container morphisms, $\eta : \mathsf{IC}(\mathsf{I}, S \triangleleft P)$, and $\mu : \mathsf{IC}((S \triangleleft P) \otimes (S \triangleleft P), S \triangleleft P)$:

$$\eta_i\ \star := (\mathsf{e}_i, P\mathsf{e}_{\equiv i})$$
$$\mu_i\ (s, s') := (s \bullet_i s', \mu_i^P(s, s'))$$

where

$$(\mu_i^P(s, s'))_k\ p := (\uparrow p, \nwarrow p, \nearrow p)$$

Showing that they satisfy the monoid equations, and that this mapping is a bijection, is mechanical and is better left to the accompanying formalization. However, intuitively, data is just being rearranged between the two types, and there is not much leeway for lossy operations. As one can imagine, the equations in the definition of ICMS are tailored so that η and μ satisfy the monoid equations. □

The Agda formalization of Lemma 3.2 requires manipulating and reasoning about the heterogeneous equalities in the definition of ICMS, which is a non-trivial exercise in intensional type theory. We direct the interested reader to our Agda formalization for all the details.

After working out the proof, we were glad to observe that the theorem is true also when the container $S \triangleleft P$ is valued in types with an arbitrary h-level, *i.e.* when $S\ i$ is not necessarily a set (what is called an h-set in homotopy type theory), and the same for $P_i\ s\ j$. The assumption of containers with Set-valued shapes and positions is not instrumental in this case. This suggests that the combinatorial specification of indexed monad containers in Definition 3.3, as well as Lemma 3.2, can be adapted to higher dimensional settings, with sets replaced by higher groupoids, which is an avenue of research that we plan to investigate in the future (more on this in the conclusive section).

Definition 3.4 (ICMS morphism ☞)**.** Given two containers $(S \triangleleft P)$ and $(S' \triangleleft P')$ equipped with structures $(\mathsf{e}, \bullet, P\mathsf{e}_{\equiv}, \uparrow, \nwarrow, \nearrow) : \mathsf{ICMS}(S \triangleleft P)$ and $(\mathsf{e}', \bullet', P\mathsf{e}'_{\equiv}, \uparrow', \nwarrow', \nearrow') : \mathsf{ICMS}(S' \triangleleft P')$, and a container morphism $f : \mathsf{IC}_I(S \triangleleft P, S' \triangleleft P')$. We say that f is an ICMS morphism between the two structures when it satisfies the following, for any index $i : I$, families of shapes $s : S\ i, s' : P_i\ s \to^I S$, index $j : I$ and position $p : P_i((\sigma\ f)_i\ (s \bullet v))\ j$:

$$(\sigma\ f)_i \equiv \mathsf{e}'_i \qquad \text{(hom-e)}$$

$$\pi\ f\ \mathsf{e}_i ;^I P\mathsf{e}_{\equiv} \equiv P\mathsf{e}'_{\equiv} \qquad \text{(hom-}P\mathsf{e}_{\equiv}\text{)}$$

$$(\sigma\ f)_i\ s \bullet' (\pi\ f ;^I v ;^I \sigma\ f) \equiv (\sigma\ f)_i\ (s \bullet v) \qquad \text{(hom-}\bullet\text{)}$$

$$\uparrow' p \equiv \uparrow(\pi\ f\ p) \qquad \text{(hom-}\uparrow\text{)}$$

$$\pi\ f(\nwarrow' p) \equiv \nwarrow(\pi\ f\ p) \qquad \text{(hom-}\nwarrow\text{)}$$

$$\pi\ f(\nearrow' p) \equiv \nearrow(\pi\ f\ p) \qquad \text{(hom-}\nearrow\text{)}$$

Such equations are obtained by applying the decomposition in Definition 2.3 to the diagrams in Definition 3.2, and in fact:

Lemma 3.3 (Characterization of monoid morphisms)**.** *Let $(S \triangleleft P, \eta, \mu)$ and $(S' \triangleleft P', \eta', \mu')$ be objects of $\mathsf{Mon}(\mathsf{IC}_I)$. A morphism $f : \mathsf{IC}_I(S \triangleleft P, S' \triangleleft P')$ is a monoid morphism between $(S \triangleleft P, \eta, \mu)$ and $(S' \triangleleft P', \eta', \mu')$ if and only if it is an* ICMS *morphism between $E(S \triangleleft P, \eta, \mu)$ and $E(S' \triangleleft P', \eta', \mu')$, where E is the equivalence in Lemma 3.2.*

Proof ☞. For a proof, we refer the reader to the formalization. □

By virtue of this equivalence, the class of ICMS morphisms is closed under identities and composition, as is that of monoid morphisms wrt. $(\mathsf{IC}_I, \mathsf{I}, \otimes)$. Hence, the following definition is well-posed.

Definition 3.5 (The category ICMSCat)**.** We define ICMSCat to be the category having indexed containers equipped with an ICMS as objects, and compatible ICMS morphisms as morphisms.

Theorem 3.4 ($\mathsf{ICMSCat} \simeq \mathsf{Mon}(\mathsf{IC}_I)$)**.** *The category of monoids in the monoidal category $(\mathsf{IC}_I, \mathsf{I}, \otimes)$ is equivalent to* ICMSCat.

Proof. It follows from the definition of ICMSCat, together with Lemma 3.2 and Lemma 3.3. □

We can now state the main result of this work:

Theorem 3.5. *The full subcategory of the category of Set^I-monads, whose endofunctors are extents of indexed containers, is equivalent to* ICMSCat.

Proof. $[\![-]\!]$ is full, faithful (Lemma 2.1) and strong monoidal (Lemma 2.3), so by Lemma 3.1, $\mathsf{Mon}(\mathsf{IC}_I)$ defines a full subcategory of $\mathsf{Mnd}(\mathsf{Set}^I) := \mathsf{Mon}(\mathsf{Endo}(\mathsf{Set}^I))$. This full subcategory is equivalent to ICMSCat by Theorem 3.4. □

4 Examples

In this section we collect a few examples of monads arising from indexed containers, as prescribed by Theorem 3.5. We show that pairs of indexed containers $S \triangleleft P$ and combinatorial monoid structures in $\mathsf{ICMS}(S \triangleleft P)$ are closed under products. We then describe indexed variants of state and writer monads. We conclude with an example of a free monad that, when appropriately instantiated, produces a type of well-scoped untyped λ-terms.

4.1 Product of two monads

Given two monads (T, η, μ) and (T', η', μ') on Set^I, one can endow the product endofunctor $T \times T'$, given by $(T \times T')\ X := TX \times T'X$, with a monad structure. Furthermore, whenever T and T' are the extents of indexed containers, their product is also the extent of an indexed container. We use our characterization to describe the induced monad structure.

Definition 4.1 (Product indexed container)**.** Let $S^0 \triangleleft P^0$ and $S^1 \triangleleft P^1$ be indexed containers. Their product is defined as $S \triangleleft P$ where

$$S\ i := S^0\ i \times S^1\ i \qquad\qquad P_i\ (s, s')\ j := P_i^0\ s\ j + P_i^1\ s'\ j$$

Suppose now that the two containers come with structure $(\mathsf{e}^0, \bullet^0, P\mathsf{e}^0_{\equiv}, \uparrow^0, \nwarrow^0, \nearrow^0) : \mathsf{ICMS}(S^0 \triangleleft P^0)$ and $(\mathsf{e}^1, \bullet^1, P\mathsf{e}^1_{\equiv}, \uparrow^1, \nwarrow^1, \nearrow^1) : \mathsf{ICMS}(S^1 \triangleleft P^1)$. We can define $(\mathsf{e}, \bullet, P\mathsf{e}_{\equiv}, \uparrow, \nwarrow, \nearrow) : \mathsf{ICMS}(S \triangleleft P)$ as follows:

$$\begin{aligned}
\mathsf{e}_i &:= (\mathsf{e}_i^0, \mathsf{e}_i^1) \\
(s_0, s_1) \bullet s' &:= (s_0 \bullet^0 \mathsf{left}\, s', s_1 \bullet^1 \mathsf{right}\, s') \\
P\mathsf{e}_{\equiv}(\mathsf{inl}\, p) &:= P\mathsf{e}^0_{\equiv}\ p \\
P\mathsf{e}_{\equiv}(\mathsf{inr}\, p) &:= P\mathsf{e}^1_{\equiv}\ p \\
\uparrow &:= [\uparrow^0, \uparrow^1] \\
\nwarrow &:= \nwarrow^0 + \nwarrow^1 \\
\nearrow &:= \nearrow^0 + \nearrow^1
\end{aligned}$$

Here $[-,-]$ is the copairing function out of a sum type, $\mathsf{left} : P_i\ (s_0, s_1) \to^I S$ is defined as $(\mathsf{left}\, s')_j := \mathsf{inl}\ ;\ s'_j\ ;\ \mathsf{fst}$, and $\mathsf{right}\, s'$ is similar with inr and snd in place of inl and fst. This definition satisfies the equations in Definition 3.3 directly by β-equivalence. The explicit witnesses can be found in the accompanying formalization.

4.2 Indexed state monad

Given a set E, the endofunctor $\mathsf{St}X := E \to E \times X$ has a canonical monad instance:

$$\begin{aligned}
&\eta_X : X \to (E \to E \times X) \\
&\eta_X\ x\ e := (e, x) \\
&\mu_X : (E \to E \times (E \to E \times X)) \to (E \to E \times X) \\
&\mu_X\ f\ e := \mathsf{let}\, (e', f') = f\ e\ \mathsf{in}\ f'\ e'
\end{aligned}$$

This is known to functional programmers as the *state monad* with state E. As described in [24], one can observe that:

$$\begin{aligned}
&E \to E \times X \\
\simeq &(E \to E) \times (E \to X) \\
\simeq &\sum_{_ : E \to E} E \to X
\end{aligned}$$

which is precisely the extent of the non-indexed container with $E \to E$ as type of shapes and E as the type of positions in each shape.

By considering a collection of sets of states $E : \mathsf{Set}^I$ instead of a single set $E : \mathsf{Set}$, we can define an indexed variant of the state monad. Given a family of sets $E : \mathsf{Set}^I$, the *indexed state endofunctor* is $\mathsf{St}^I_E\ X\ i := E\ i \to \sum_{j:I} E\ j \times X\ j$.

Conceptually, we can describe an indexed stateful computation as follows: I is a set of *modes* that our stateful computer can be in. Not all states are available in every mode, so the set of states is replaced by a collection of sets $E\ i$ containing the states available in mode i. From every mode i and state $\mathsf{e} : E\ i$, a stateful computation, besides returning the resulting value from $X\ i$, points the computer to a new mode $j : I$, and sets a new state in $E\ j$, in which the next stateful computation can be performed.

By a similar reasoning as in the non-indexed case above, we can construct the following chain of equivalence.

$$
\begin{aligned}
& E\ i \to \sum_{j:I} E\ j \times X\ j \\
\simeq & \sum_{m:E\,i \to I} \prod_{e:E\,i} E\ (m\ e) \times X\ (m\ e) && \text{(Σ-Π interchange)} \\
\simeq & \sum_{m:E\,i \to I} \left(\prod_{e:E\,i} E\ (m\ e) \right) \times \left(\prod_{e:E\,i} X\ (m\ e) \right) && \text{(Universal property of product)} \\
\simeq & \sum_{(m,\_):\sum_{m:E\,i \to I} (\prod_{e:E\,i} E\,(m\,e))} \left(\prod_{e:E\,i} X\ (m\ e) \right) && \text{(associativity of Σ)} \\
\simeq & \sum_{s:E\,i \to \sum_{j:I} E\,j} \left(\prod_{e:E\,i} X\ \mathsf{fst}(s\ e) \right) && \text{(Σ-Π interchange)} \\
\simeq & \sum_{s:E\,i \to \sum_{j:I} E\,j} \left(\prod_{j:I} \prod_{e:E\,i} \mathsf{fst}(s\ e) \equiv j \to X\ j \right) && \text{(Yoneda)} \\
\simeq & \sum_{s:E\,i \to \sum_{j:I} E\,j} \left(\prod_{j:I} \left(\sum_{e:E\,i} \mathsf{fst}(s\ e) \equiv j \right) \to X\ j \right) && \text{(currying)}
\end{aligned}
$$

The latter type is the extent of the container $S \triangleleft P$, with:

$$
S\ i := E\ i \to \sum_{j:I} E\ j \qquad\qquad P_i\ s\ j := \sum_{e:E\,i} \mathsf{fst}(s\ e) \equiv j
$$

A monoid structure in $\mathsf{ICMS}(S \triangleleft P)$ can be defined as follows:

$$
\begin{aligned}
\mathsf{e}_i\ e &:= (i, e) \\
s \bullet s' &:= \lambda e.\ \mathsf{let}\ (j, e') = s\ e\ \mathsf{in}\ s'_j\ (e, \mathsf{refl})\ e' \\
P\mathsf{e}_{\equiv} &:= \mathsf{snd} \\
\uparrow_{i,s,\_,j}(e, \_) &:= \mathsf{fst}(s\ e) \\
\nwarrow_{i,s,\_,j}(e, \_) &:= (e, \mathsf{refl}) \\
\nearrow_{i,s,\_,j}(e, p) &:= (\mathsf{snd}(s\ e), p)
\end{aligned}
$$

When $I = \mathbb{1}$, the extent of the container becomes isomorphic to the usual state monad endofunctor, and the same is true for the monad structure. The associativity-related equations in Definition 3.3 hold definitionally, while the unit-related ones can be proved via functional extensionality by manipulating the equalities in the hypotheses. For further details, please refer to the formalization.

4.3 Indexed writer monad

Given a Set-monoid $(W, \varepsilon, (\cdot))$, the endofunctor $\mathsf{Wr}\ X := W \times X$ can be given a monad structure defined by:

$$\begin{aligned}
&\eta_X : X \to W \times X \\
&\eta_X\ x := (\varepsilon, x) \\
&\mu_X : W \times (W \times X) \to W \times X \\
&\mu_X\ (w, (w', x)) := (w \cdot w', x)
\end{aligned}$$

It satisfies the monad equations by virtue of $(W, \varepsilon, (\cdot))$ being a monoid. This is usually referred to as the *writer* monad, and is clearly an instance of a container monad arising from the container $W \lhd (\lambda\_.\mathbb{1})$. One can immediately generalise the above to the Set^I-endofunctor $F\ X\ i := W \times X\ i$, essentially obtaining an I-indexed product of writer monads.

However, we can go a step further, allowing the endofunctor to change indices in a way that suits the monoid structure on W. Recall that an *action* of W on I is a monoid homomorphism from $(W, \varepsilon, (\cdot))$ to the monoid of endofunctions $((I \to I), \mathsf{id}_I, ;)$. Let $\blacktriangleright$ be such an action and define the endofunctor

$$\mathsf{Wr}^{\blacktriangleright}\ X\ i := \sum_{w:W} X\ (w \blacktriangleright i)$$

Note that when we pick the trivial action $(w \blacktriangleright i := i)$, we get back the I-indexed product of writer monads described above. The endofunctor $\mathsf{Wr}^{\blacktriangleright}$ is isomorphic to the extent of the container $S \lhd P$ given by:

$$S\ i := W \qquad\qquad P_i\ w\ j := w \blacktriangleright i \equiv j$$

We can then adapt the usual writer monad structure to this heterogeneously indexed variant in the following way:

$$\begin{aligned}
\mathsf{e}_i &:= \varepsilon \\
w \bullet w' &:= w \cdot w' \\
P\mathsf{e}_{\equiv} &:= (\blacktriangleright \text{ preserves } \varepsilon) && \text{(witnesses } \varepsilon \blacktriangleright i \equiv j \to i \equiv j\text{)} \\
\uparrow_{i,w,_,j}\ _ &:= w \blacktriangleright i \\
\nwarrow_{i,w,_,j}\ _ &:= \mathsf{refl} \\
\nearrow_{i,w,_,j}\ _ &:= (\blacktriangleright \text{ preserves } (\cdot)) && \text{(witnesses } (w \cdot w') \blacktriangleright i \equiv j \to w' \blacktriangleright (w \blacktriangleright i) \equiv j\text{)}
\end{aligned}$$

Notice that the purpose of the set of positions $P_i\ w\ j$ is to express dependency constraints between indices i and j through the action $\blacktriangleright$. Therefore, in this ICMS, the components on shapes are the same that appear in the non-indexed case, while the components on positions witness the fact that the monad unit and multiplication respect the indexing constraints.

Lastly, the only data-relevant equations that need to be fulfilled are Equations (e-unit-l), (e-unit-r) and (•-assoc), which are granted directly by the monoid structure on W.

4.4 Well-scoped λ-terms

Given an indexed container $S^0 \lhd P^0$, it is possible to construct a new container $S \lhd P$ and an element of $\mathsf{ICMS}(S \lhd P)$ such that $[\![S \lhd P]\!]$ is the free monad on the endofunctor $[\![S^0 \lhd P^0]\!]$. We do not present in the paper the general construction for building the free indexed container monoid structure on an arbitrary indexed container $S^0 \lhd P^0$. We only show a particular example which illustrates the construction clearly.

In their work on indexed containers, Altenkirch et al. [6, Introduction] point out that a family of well-scoped λ-terms can be defined in dependent type theory as the initial algebra for the endofunctor on $\mathsf{Set}^{\mathsf{Nat}}$ given by $X \mapsto \lambda n.\ \mathsf{Fin}\ n + (X\ n)^2 + X\ (n+1)$, where $\mathsf{Fin}\ n := \{0, 1, \ldots, n-1\}$.

As reported by Gambino and Hyland [12], given a polynomial endofunctor F on a locally Cartesian closed category, the free monad F^* on F can be constructed as the functor taking an object Y to the initial algebra of the endofunctor $X \mapsto Y + F\ X$, whenever such initial algebra exists. The resulting endofunctor F^* is also polynomial and an appropriate monad structure on F^* can be canonically built.

We can perform this construction for the endofunctor F on $\mathsf{Set}^{\mathsf{Nat}}$ given by $F\ X\ n := (X\ n)^2 + X\ (n+1)$ and obtain $F^*\ Y\ n := \mu X.\ Y\ n + (X\ n)^2 + X\ (n+1)$ as the carrier of the free monad on F. Instantiating Y to Fin gives a collection of types $F^*\,\mathsf{Fin}$, which is the same family of λ-terms described above.

Notice that the functor F is equivalent to the extent of the container $S^0 \triangleleft P^0$, with $S^0\ n := \mathsf{Bool}$ and

$$P^0_n\ \mathsf{false}\ m := \mathsf{Bool} \times (n \equiv m)$$
$$P^0_n\ \mathsf{true}\ m := (n+1 \equiv m)$$

The functor F^* can also be equivalently presented as the extent of an indexed container $[\![S \triangleleft P]\!]$. Shapes can be given as an indexed W-type S, that we describe as an inductive type family using Agda-like syntax.

$$\begin{aligned}
&\mathsf{data}\ S\ (n : \mathsf{Nat}) : \mathsf{Set}\ \mathsf{where}\\
&\quad \mathsf{var} : S\ n\\
&\quad \mathsf{app} : S\ n \to S\ n \to S\ n\\
&\quad \mathsf{lam} : S\ (n+1) \to S\ n
\end{aligned}$$

The family of positions P is defined by recursion on the first shape in $S\ n$ as follows:

$$\begin{aligned}
P_n\ \mathsf{var}\ m &:= (n \equiv m)\\
P_n\ (\mathsf{app}\ M\ N)\ m &:= P_n\ M\ m + P_n\ N\ m\\
P_n\ (\mathsf{lam}\ M)\ m &:= P_{n+1}\ M\ m
\end{aligned}$$

One can then verify that $[\![S \triangleleft P]\!]$ satisfies the fixpoint equation characterizing F^*. The free monad structure is given as an element of $\mathsf{ICMS}(S \triangleleft P)$. The operations $\bullet$, $\uparrow$, $\nwarrow$ and $\nearrow$ are defined by structural recursion on the first given shape in $S\ n$. The required equations are proved by structural induction.

$$\begin{aligned}
\mathsf{e}_n &:= \mathsf{var}\\
\mathsf{var} \bullet \sigma &:= \sigma\ \mathsf{refl}\\
(\mathsf{app}\, M\ N) \bullet \sigma &:= \mathsf{app}\,(M \bullet (\mathsf{inl}; \sigma))(N \bullet (\mathsf{inr}; \sigma))\\
(\mathsf{lam}\, M) \bullet \sigma &:= \mathsf{lam}\,(M \bullet \sigma)\\
P\mathsf{e}_{\equiv}\ p &:= p\\
\uparrow_{n,\mathsf{var},\sigma,m\,-} &:= n\\
\uparrow_{n,\mathsf{app}\,M\,N,\sigma,m}(\mathsf{inl}\ p) &:= \uparrow_{n,M,(\mathsf{inl};\sigma),m} p\\
\uparrow_{n,\mathsf{app}\,M\,N,\sigma,m}(\mathsf{inr}\ p) &:= \uparrow_{n,N,(\mathsf{inr};\sigma),m} p\\
\uparrow_{n,\mathsf{lam}\,M,\sigma,m} p &:= \uparrow_{n+1,M,\sigma,m} p\\
\nwarrow_{n,\mathsf{var},\sigma,m\,-} &:= \mathsf{refl}\\
\nwarrow_{n,\mathsf{app}\,M\,N,\sigma,m}(\mathsf{inl}\ p) &:= \mathsf{inl}(\nwarrow_{n,M,(\mathsf{inl};\sigma),m} p)
\end{aligned}$$

$$\begin{aligned}
\nwarrow_{n,\mathsf{app}MN,\sigma,m}(\mathsf{inr}\,p) &:= \mathsf{inr}(\nwarrow_{n,N,(\mathsf{inr};\sigma),m}p)\\
\nwarrow_{n,\mathsf{lam}M,\sigma,m}p &:= \nwarrow_{n+1,M,\sigma,m}p\\
\nearrow_{n,\mathsf{var},\sigma,m}p &:= p\\
\nearrow_{n,\mathsf{app}MN,\sigma,m}(\mathsf{inl}\,p) &:= \nearrow_{n,M,(\mathsf{inl};\sigma),m}p\\
\nearrow_{n,\mathsf{app}MN,\sigma,m}(\mathsf{inr}\,p) &:= \nearrow_{n,N,(\mathsf{inr};\sigma),m}p\\
\nearrow_{n,\mathsf{lam}M,\sigma,m}p &:= \nearrow_{n+1,M,\sigma,m}p
\end{aligned}$$

We remark that, while F^* is a monad on $\mathsf{Set}^{\mathsf{Nat}}$, the functor $F^*\,\mathsf{Fin} : \mathsf{Nat} \to \mathsf{Set}$ is a monad relative to the functor $\mathsf{Fin} : \mathsf{Nat} \to \mathsf{Set}$ and this latter relative monad structure is more fundamental for well-scoped λ-terms that the former monad structure. In particular, substitution of well-scoped λ-terms is described by the multiplication of $F^*\,\mathsf{Fin}$, not the multiplication of F^*.

5 Conclusions

In this paper we performed a combinatorial analysis of monad structures on the extent of indexed containers, extending previous work by Uustalu on the non-indexed case. The main technical contribution is a combinatorial characterization of monoid structures in the monoidal category of indexed containers, with the identity container and container composition as monoidal unit and tensor. With this characterization, we generate many examples of interest in dependently-typed functional programming, such as indexed extensions of the state and the writer monad and a free monad producing the type of well-scoped λ-terms. Our combinatorial analysis should serve as a tool for easing the construction of monad structures on certain endofunctors, for disproving that a candidate monad structure can possibly work, or for enumerative purposes.

Uustalu [24] also discusses conditions ensuring that the resulting monads are Cartesian. We should be able to easily generalize these conditions to the indexed case.

When performing the formalization, we noticed that although our theoretical setting is based on Set-families, many of the constructions presented in the paper work for types with arbitrary h-level, not just sets. In particular, there is no global assumption of Uniqueness of Identity Proofs. The restriction to sets is necessary in the proof of Lemma 2.1 (it is more generally needed to show that indexed containers form a category, where homtypes are required to be sets) and also employed in the formalization of examples. In future work, we intend to extend our analysis to higher-categorical dimensions, moving from categories of (indexed) *sets* to higher categories of (indexed) *higher groupoids*. The case of pseudofunctors and pseudomonads on 1-groupoids, a.k.a. types with h-level 1 in the sense of homotopy type theory [23], is already of interest. Kock [18] showed that polynomial functors on groupoids capture datatypes with symmetries, which are closely connected to analytic functors and Joyal's combinatorial species [16]. *E.g.* a groupoid of finite multisets can be defined as the extent of a non-indexed container with shapes given by the "groupoid of finite types" and positions at a finite type given by its cardinality [11, 14].

A different avenue of future works concerns the use of indexed containers in the syntax and semantics of higher inductive types in HoTT. Van der Weide and Geuvers [27] introduce a schema for the specification of finitary set-truncated HITs, which are semantically interpreted as set-quotients of some initial algebras, and analogous studies exist for finitary 1-truncated HITs [10, 25]. The expressivity of these schemata could be abundantly enhanced by moving to a notion of signature based on indexed containers.

Acknowledgments We are grateful to our reviewers for the numerous helpful comments they made. M.D.P. and N.V. were supported by the Estonian Research Council grant no. PSG749. T.U. was supported by the Estonian Research Countil grant no. PRG1210.

References

[1] Michael Abbott, Thorsten Altenkirch & Neil Ghani (2005): *Containers: Constructing Strictly Positive Types*. *Theoretical Computer Science* 342(1), pp. 3–27, doi:10.1016/j.tcs.2005.06.002.

[2] C. B. Aberlé & David I. Spivak (2024): *Polynomial Universes and Dependent Types*. arXiv eprint arXiv:2409.19176 [cs.LO], doi:10.48550/arxiv.2409.19176.

[3] Danel Ahman, James Chapman & Tarmo Uustalu (2014): *When Is a Container a Comonad?* *Log. Methods in Comput. Sci.* 10(3), pp. 14:1–14:48, doi:10.2168/lmcs-10(3:14)2014.

[4] Danel Ahman & Tarmo Uustalu (2013): *Distributive Laws of Directed Containers*. *Progress in Informatics* 10, pp. 3–18, doi:10.2201/niipi.2013.10.2.

[5] Danel Ahman & Tarmo Uustalu (2014): *Update Monads: Cointerpreting Directed Containers*. In Ralph Matthes & Aleksy Schubert, editors: *19th Int. Conf. on Types for Proofs and Programs, TYPES 2013*, *Leibniz Int. Proc. in Informatics* 26, Dagstuhl Publishing, pp. 1:1–1:23, doi:10.4230/lipics.types.2013.1.

[6] Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride & Peter Morris (2015): *Indexed Containers*. *Journal of Functional Programming* 25, pp. e5:1–e5:41, doi:10.1017/s095679681500009x.

[7] Steve Awodey (2018): *Natural Models of Homotopy Type Theory*. *Math. Struct. Comput. Sci.* 28(2), pp. 241–286, doi:10.1017/s0960129516000268.

[8] Steve Awodey & Clive Newstead (2018): *Polynomial Pseudomonads and Dependent Type Theory*. arXiv eprint 1802.00997 [math.CT], doi:10.48550/arxiv.1802.00997.

[9] Stefania Damato, Thorsten Altenkirch & Axel Ljungström (to appear): *Formalising Inductive and Coinductive Containers*. In Yannick Forster & Chantal Keller, editors: *16th Int. Conf. on Interactive Theorem Proving, ITP 2025*, Leibniz Int. Proc. in Informatics, Dagstuhl Publishing.

[10] Peter Dybjer & Hugo Moeneclaey (2018): *Finitary Higher Inductive Types in the Groupoid Model*. *Electron. Notes in Theor. Comput. Sci.* 341, pp. 119–134, doi:10.1016/j.entcs.2018.03.019.

[11] Eric Finster, Samuel Mimram, Maxime Lucas & Thomas Seiller (2021): *A Cartesian Bicategory of Polynomial Functors in Homotopy Type Theory*. In Ana Sokolova, editor: *Proc. of 37th Conf. on Mathematical Foundations of Programming Semantics, MFPS 2021*, *Electron. Proc. in Theor. Comput. Sci.* 351, Open Publishing Assoc., pp. 67–83, doi:10.4204/eptcs.351.5.

[12] Nicola Gambino & Martin Hyland (2004): *Wellfounded Trees and Dependent Polynomial Functors*. In Stefano Berardi, Mario Coppo & Ferrucio Damiani, editors: *Types for Proofs and Programs, TYPES 2003*, *Lect. Notes in Comput. Sci.* 3085, Springer, pp. 210–225, doi:10.1007/978-3-540-24849-1_14.

[13] Nicola Gambino & Joachim Kock (2013): *Polynomial Functors and Polynomial Monads*. *Math. Proc. Cambridge Phil. Soc.* 154(1), pp. 153–192, doi:10.1017/s0305004112000394.

[14] Philipp Joram & Niccolò Veltri (2023): *Constructive Final Semantics of Finite Bags*. In Adam Naumowicz & René Thiemann, editors: *14th Int. Conf. on Interactive Theorem Proving, ITP 2023*, *Leibniz Int. Proc. in Informatics* 268, Dagstuhl Publishing, pp. 20:1–20:19, doi:10.4230/lipics.itp.2023.20.

[15] Philipp Joram & Niccolò Veltri (2025): *Data Types with Symmetries via Action Containers*. In Rasmus Ejlers Møgelberg & Benno van der Berg, editors: *30th Int. Conf. on Types for Proofs and Programs, TYPES 2024*, *Leibniz Int. Proc. in Informatics* 336, Dagstuhl Publishing, pp. 6:1–6:21, doi:10.4230/lipics.types.2024.6.

[16] André Joyal (1986): *Foncteurs analytiques et espèces de structures*. In Gilbert Labelle & Pierre Leroux, editors: *Combinatoire énumérative*, *Lect. Notes in Math.* 1234, Springer, pp. 126–159, doi:10.1007/bfb0072514.

[17] André Joyal & Ross Street (1993): *Braided Tensor Categories*. *Advances in Mathematics* 102(1), pp. 20–78, doi:10.1006/aima.1993.1055.

[18] Joachim Kock (2012): *Data Types with Symmetries and Polynomial Functors over Groupoids*. *Electron. Notes Theor. Comput. Sci.* 286, pp. 351–365, doi:10.1016/j.entcs.2013.01.001.

[19] Fosco Loregian (2021): *(Co)end Calculus*. *London Math. Soc. Lect. Note Series* 468, Cambridge University Press, doi:10.1017/9781108778657.

[20] Saunders Mac Lane (1978): *Categories for the Working Mathematician*, 2nd edition. *Graduate Texts in Mathematics* 5, Springer, doi:10.1007/978-1-4757-4721-8.

[21] Clive Newstead (2018): *Algebraic Models of Dependent Type Theory*. Ph.D. thesis, Carnegie Mellon University, doi:10.1184/r1/7195124.v1.

[22] Chris Purdy & Stefania Damato (to appear): *Distributive Laws of Monadic Containers*. In Corina Cirstea & Alexander Knapp, editors: *11th Conf. on Algebra and Coalgebra in Computer Science, CALCO 2025*, Leibniz Int. Proc. in Informatics, Dagstuhl Publishing.

[23] The Univalent Foundations Program (2013): *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study. Available at `https://homotopytypetheory.org/book`.

[24] Tarmo Uustalu (2017): *Container Combinatorics: Monads and Lax Monoidal Functors*. In Mohammad Reza Mousavi & Jiří Sgall, editors: *Topics in Theoretical Computer Science, TTCS 2017*, *Lecture Notes in Computer Science* 10608, Springer, pp. 91–105, doi:10.1007/978-3-319-68953-1_8.

[25] Niccolò Veltri & Niels van der Weide (2021): *Constructing Higher Inductive Types as Groupoid Quotients*. *Log. Methods Comput. Sci.* 17(2), pp. 8:1–8:42, doi:10.23638/lmcs-17(2:8)2021.

[26] Mark Weber (2015): *Polynomials in Categories with Pullbacks*. *Theor. Appl. Categ.* 30(16), pp. 533–598. Available at `http://www.tac.mta.ca/tac/volumes/30/16/30-16abs.html`.

[27] Niels van der Weide & Herman Geuvers (2019): *The Construction of Set-Truncated Higher Inductive Types*. *Electron. Notes Theor. Comput. Sci.* 347, pp. 261–280, doi:10.1016/j.entcs.2019.09.014.

Nominal Sets in Rocq

Fabrício Sanches Paranhos
Nethermind
London, UK
fabricio@nethermind.io

Daniel Ventura
Univ. Federal de Goiás
Goiânia, Brazil
ventura@ufg.br

Nominal techniques have been praised for their ability to formalize grammars with binding structures closer to their informal developments. At its core, there lies the definition of nominal sets, which capture the notion of name (in)dependence through a simple, and uniform, metatheory based on name permutations. We present a formal constructive development of nominal sets in Rocq (formerly known as Coq), with its main design and project decisions. Furthermore, we formalize the concepts of freshness, nominal α-equivalence, name abstraction, and finitely supported functions. Our implementation relies on a type class hierarchy which, combined with Rocq generalized rewriting mechanism, achieves concise definitions and proofs, whilst easing the well-known "setoid hell" scenario. We conclude with a discussion on how to obtain the constructive α-structural recursion and induction combinators, towards a nominal framework.

1 Introduction

Nominal techniques [21] have been praised due to its capacity to formalize syntax with binders closer to their informal developments (pen and paper proofs) [10, 14]. However, given the elaborate metatheory of nominal sets [20] and the lack of a formal development of an infrastructure (cf. [28]), many nominal-based formalization [3, 4, 5, 6, 13, 14, 15, 29], named pragmatic here, choose to apply the bare minimum ideas from nominal logic to a specific issue, *e.g.* in a formalization of the λ-calculus and the π-calculus.

The consequence of a pragmatic approach is that, whenever a new development chooses to use nominal logic, the whole infrastructure needs to be rebuilt. Nominal frameworks, like Isabelle's nominal package [28], provide the necessary infrastructure and automation, which can reduce the amount of man-hour considerably. Unfortunately, to the best of our knowledge, no other proof assistant has such a framework. In particular, there is a proof of concept in Rocq (formerly known as Coq) [6], with no further development, that presents an approach based on axiomatization. In [1], Pitts mentions the lack of support for nominal techniques in dependent-type based systems, such as Rocq and Agda, whilst posing a question about the possibility of a dependent type theory corresponding, under the Curry-Howard Correspondence, to a constructive nominal logic that could be useful as a nominal framework. In [12] such a possibility is investigated in an Agda formal development, which relies heavily on setoids, and formalizes the core aspects of the nominal theory: support, freshness and name abstraction.

We propose a formalization of constructive nominal sets in Rocq, a foundation towards a nominal framework, following the approach in [12] closely. Nominal sets modelled as a typeclass hierarchy was already introduced in [18] and, besides formalization of the core of the nominal theory, we specified two notions related to the derivation of α-structural principles: the freshness theorem and the freshness condition for binders (FCB). We claim that by using Rocq's typeclass [24] and the generalized rewriting mechanism [23], we achieve much shorter and simpler proofs, with some of them close resembling its informal, pen and paper, counterpart.

H. Barbosa, C. Ringeissen (Eds.): 20th International Symposium
on Logical and Semantic Frameworks with Applications
EPTCS 430, 2025, pp. 55–68, doi:10.4204/EPTCS.430.5

The remaining text is structured as follows: in Section 2 we introduce the (constructive) nominal sets and related notions; in Section 3 we discuss some aspects of the Rocq specification; and in Section 4 we conclude with related and future work.

Full development, using Rocq version 8.20.1, can be found at `https://github.com/fasapa/nominal/tree/lsfa2025`.

2 Constructive Nominal Sets

Nominal sets capture the notion of name (in)dependence through a simple, and uniform, metatheory based on name permutations. The **set of names** $\mathbb{A}$ is an infinite set of **atoms**, where the internal structure of its terms is irrelevant and the only relevant attribute is the identity relation. By enriching a structure built from $\mathbb{A}$ with (left group) action of permutations such a notion of name dependence can be defined, where the set of names that a structure depends on is called **support** while its complement is formed by the so-called **fresh names**. In other words, nominal sets give us the tools to express name dependence for any of its elements by means of name permutation only. In what follows, we present the key notions in order to define nominal sets (see [20] for further details).

Given any set A, the set $\mathscr{S}_A$ of bijections $f : A \to A$ is a well-known algebraic structure, known as the **permutation group**, when combined with function composition $\circ$. In particular, when A is infinite, the subset $P_A \subset \mathscr{S}_A$ formed only by f such that $\{x \in A \,|\, x \neq f(x)\}$ is finite, called **finite permutations**, is also a group with the same operation. $\langle P_{\mathbb{A}}, \circ \rangle$ is called the **name permutation group**, where p^{-1} denotes the inverse of $p \in P_{\mathbb{A}}$ and `id` its neutral element. Another well-known result about finite permutations[1] is that any permutation may be represented as a composition of **transpositions** of the form $(a\ b)$, also called **swapping**, mapping a to b and b to a while any other $c \in A$ is mapped to itself. Finite permutations represented as sequences of swaps is the base of the current formal development (see Sec. 3).

Let X be a set and $\langle G, * \rangle$ be a group with neutral element ε. **Left action** of G on X is a function $(\bullet) : G \times X \to X$ satisfying:

$$(\forall x \in X),\ \varepsilon \bullet x = x \qquad\qquad (\forall g, h \in G)(\forall x \in X),\ g \bullet (h \bullet x) = (h * g) \bullet x$$

Such an action is called a **permutation action** when $G = P_{\mathbb{A}}$ and X is then called a **Perm-set**.

Let X be any Perm-set, we say that a finite $S \subset \mathbb{A}$ is a (finite) **support** of some $x \in X$ if:

$$(\forall p \in P_{\mathbb{A}}), (((\forall a \in S), p \bullet a = a) \Rightarrow p \bullet x = x) \tag{1}$$

A support is then a finite collection of all names in which an element x depends on, in the sense that any permutation fixing each name in this set also fixes x. A Perm-set X is a **nominal set** if all of its elements have a finite support. For instance, $\mathbb{A}$ is itself a nominal set –where a is supported by $\{a\}$, for any $a \in \mathbb{A}$– and the boolean set $\mathbb{B}$ is the so-called **trivial nominal set** –where each element is supported by the empty set. From now on, X denotes a nominal set unless otherwise stated.

Supports are not unique and, given S_1 and S_2, two supports of some $x \in X$, $S_1 \cap S_2$ is also a support of the same element [20]. We can thus define the **least support** of an $x \in X$ as follows:

$$\mathtt{supp}(x) \triangleq \bigcap \{S \mid S \text{ supports } x\} \tag{2}$$

In other words, the least support of an element x is the all and only names in which x depends on. In a classical setting, every finitely supported element is proven to have a least support [20]. Least supports

[1] Note that for A a finite set, any permutation in $\mathscr{S}_A$ is finite.

allows to obtain a stronger statement than (1):

$$(\forall p \in P_{\mathbb{A}}), (((\forall a \in \mathtt{supp}(x)), p \bullet a = a) \iff p \bullet x = x) \tag{3}$$

Freshness relation is then defined as the complementary notion of least support, where given X and Y, nominal sets, for any $x \in X$, $y \in Y$:

$$x \# y \Leftrightarrow \mathtt{supp}(x) \cap \mathtt{supp}(y) = \emptyset \tag{4}$$

In particular, if $X = \mathbb{A}$ then $x \# y \Leftrightarrow x \notin \mathtt{supp}(y)$ and x is said to be **fresh** for y. Given y finitely supported, the set $\mathbb{A} \setminus \mathtt{supp}(y)$ of fresh names for y is infinite, thus to choose a fresh name is always possible, called **choose-a-fresh-name principle**. Freshness is an **equivariant** relation, *i.e.* closed for permutation action: $\forall p \in P_{\mathbb{A}}, x \# y \Rightarrow (p \bullet x) \# (p \bullet y)$. In particular, equivariance can be restated when $X = \mathbb{A}$ as:

$$\forall p \in P_{\mathbb{A}},\ x \notin \mathtt{supp}(y) \ \Rightarrow\ (p \bullet x) \notin \mathtt{supp}(p \bullet y) \tag{5}$$

It is worth noticing that support (and freshness) is relative to the equality considered in X. For instance, let X be the set Λ of λ-terms and $t \in \Lambda$. The support of t is the set of all the term-variables occurring in t, when the syntactic equality in Λ is considered, while it is the set of free variables when α-equivalence, with its associated equality on Λ, is considered instead.

Two key notions can be defined in a rather abstract way for nominal sets: α-equivalence and name abstraction. First, α-**equivalence** is a binary relation on $\mathbb{A} \times X$, denoted by $\approx_\alpha$, defined by:

$$(a,x) \approx_\alpha (b,y) \ \triangleq\ (\exists c \in \mathbb{A}),\ c \# (a,b,x,y) \ \wedge\ (a\ c) \bullet x = (b\ c) \bullet y \tag{6}$$

where $c \# (a,b,x,y)$ denotes $c \# a \wedge c \# b \wedge c \# x \wedge c \# y$. By the Some/Any Theorem (Thm. 3.9 in [20]) an equivalent definition can be given:

$$(a,x) \approx_\alpha (b,y) \ \triangleq\ (\forall c \in \mathbb{A}),\ c \# (a,b,x,y) \ \Rightarrow\ (a\ c) \bullet x = (b\ c) \bullet y \tag{7}$$

Such equivalences are considered in [20] with respect to the M quantifier, where $(\mathit{M}a \in \mathbb{A})\varphi$ means that φ holds for all but finitely many atoms. Changing from "some fresh" to "any fresh" name is identified as a common pattern while reasoning with fresh names [20], and the utility of having two such definitions is briefly discussed in the locally nameless approach [11], where a cofinite quantification is proposed instead[2]. We call **some/any principle** the reasoning in proving equivalences between existential/universal predicates in the formal development.

Second, the **name abstraction set** of X is the set $\mathbb{A} \times X$ modulo $\approx_\alpha$, denoted by $[\mathbb{A}]X$. The name abstraction $[a]x$ denotes the class $c \in [\mathbb{A}]X$ where $(a,x) \in c$. Observe that permutation action on $[\mathbb{A}]X$ is well-defined by $p \bullet [a]x \triangleq [p \bullet a](p \bullet x)$ and is proved that $\mathtt{supp}([a]x) = \mathtt{supp}(x) \setminus \{a\}$, for any $[a]x \in [\mathbb{A}]X$. The proof that $[\mathbb{A}]X$ is a nominal set relies on that a quotient of a nominal set by an equivariant equivalence relation is also nominal [20]: $\mathbb{A} \times X$ is nominal –product of two nominal sets– and $\approx_\alpha$ is proved to be equivariant.

Given X, Y nominal sets, let $X \to Y$ be the set of all functions from X to Y. Considering the permutation action defined by $p \bullet F \triangleq \lambda x \in X \to p \bullet (F(p^{-1} \bullet x))$, the set:

$$X \to_{\mathtt{fs}} Y \ \triangleq\ \{F \in X \to Y \mid (\exists S \subset \mathbb{A}),\ F \text{ is supported by finite } S\}$$

[2] Cofinite quantification restricted to finitely supported predicates corresponds to the M quantification [22].

of the **finitely supported functions** is equivariant[3] and is a nominal set, called **nominal function set.** Finitely supported functions are essential to obtain α-structural recursion and induction principles, as presented in [19]. Two last properties are needed to obtain a recursion combinator: the Freshness theorem and the Freshness Condition for Binders (FCB).

Theorem 2.1 (Freshness theorem [19]) *Given a nominal set X and a finitely supported function $h \in \mathbb{A} \to_{\mathtt{fs}} X$ satisfying*

$$(\exists a \in \mathbb{A}), a \# h \wedge a \# h(a)$$

then exists a unique element $\mathtt{fresh}(h) \in X$ such that

$$(\forall a \in \mathbb{A}), a \# h \Rightarrow h(a) = \mathtt{fresh}(h)$$

The theorem in [20] is stated considering partial functions and N quantification over the set of names (see Theorem 3.11, [20]). A function $f : X \to Y$ is **respectful** to $\approx_X$ and $\approx_Y$, equalities for X and Y respectively, if

$$(\forall a\ b \in X),\ a \approx_X b \Rightarrow f(a) \approx_Y f(b)$$

We say that a function **respect** equivalence relations if it is respectful to their induced equalities.

Theorem 2.2 (Freshness Condition for Binders (FCB) [20, 12]) *Given nominal sets X and Y and a finitely supported function $f \in (\mathbb{A} \times X) \to_{\mathtt{fs}} Y$ satisfying*

$$(\exists a \in \mathbb{A}), a \# f \wedge (\forall x \in X), a \# f(a,x)$$

then exists a unique function $\overline{f} \in [\mathbb{A}]X \to_{\mathtt{fs}} Y$ such that

$$(\forall a \in \mathbb{A}), a \# f \Rightarrow (\forall x \in X), \overline{f}([a]x) = f(a,x)$$

Any function related to some binder would have the signature $\mathbb{A} \times X \to_{\mathtt{fs}} Y$ and it would need to respect the nominal α-equivalence. Once such a compatibility is proved, we can lift the argument $\mathbb{A} \times X$ into $[\mathbb{A}]X$, ensuring the function is well-defined over the α-equivalence classes. FCB provides a simpler alternative, based on freshness, to this otherwise difficult approach. With FCB we can construct a well-defined function over the α-equivalence classes, with type $[\mathbb{A}]X \to_{\mathtt{fs}} Y$, from a simpler one, with type $\mathbb{A} \times X \to_{\mathtt{fs}} Y$, where a some/any principal is helpful since we would just need to show that the condition is valid for a single name.

We postpone a discussion on the α-structural recursion and induction, including the related some/any principle of freshness condition for binders (FCB), to Section 4.

Constructive Nominal Sets: Existence of least supports cannot be proved constructively [26, 27]. In fact, admitting the existence of least supports is equivalent to the non-constructive *weak limited principle of omniscience* (see [27] for a discussion). In constructive developments such as [12, 17] the least support is replaced by **"some" support**. In other words, given $x \in X$, S_x is any set satisfying (1) which can also be seen as an upper bound for $\mathtt{supp}(x)$ since $\mathtt{supp}(x) \subseteq S_x$, for any such a set. A characterisation of support based on swaps is given in [20], and used in [12], where S_x is any S satisfying:

$$(\forall a, b \in \mathbb{A} \setminus S), (a\ b) \bullet x = x \tag{8}$$

[3] If F is supported by S then $p \bullet F$ is supported by $\{p(a) \mid a \in S\}$.

Notions such as freshness need to be addressed while considering some instead of least support. For instance, $a \notin S_x \Rightarrow a \# x$ for any S_x but $a \# x \not\Rightarrow a \notin S_x$, since S_x is not unique. Therefore, equivariance of the freshness relation cannot be stated as in (5) since $a \notin S_x \not\Rightarrow (p \bullet a) \notin S_{(p \bullet x)}$, for some $p \in P_{\mathbb{A}}$. **Constructive freshness** is then defined in [12] by:

$$a \# x \triangleq (\exists b \in \mathbb{A}),\ b \notin S_x \wedge (a\ b) \bullet x = x \tag{9}$$

and a dual definition, proved to be equivalent by a some/any principle:

$$a \# x \triangleq (\forall b \in \mathbb{A}),\ b \notin S_x \Rightarrow (a\ b) \bullet x = x \tag{10}$$

The relation is proved to be equivariant [12] and α-equivalence can then be defined as in (6-7). It is worth noticing that a Some/Any Theorem cannot be proved in the current development[4] thus equivalences between universal and existential formulas are proved instead, referred to as **some/any principles**. They are especially important for a nominal framework, since these some/any principles allow us to generalize a proof of a property about a single name to any name. Proofs of such a principle in [12] follows the reasoning in Lemma 3.7 from [20], used to prove the Some/Any Theorem. The present development follows the same approach. Name abstraction is defined as before, with a different proof (see Section 3).

When considering syntactic equality as the equivalence relation, the formalization of the nominal function set demands the axiom of functional extensionality. To avoid adding the extensionality axiom, Choudhury worked with setoids in [12], which quickly became, in his words, "unwieldy". To escape the "setoid hell" scenario [2], we leverage Rocq's typeclass together with typeclass based generalized rewriting mechanism, and model nominal sets as an unbundled typeclass hierarchy [18, 25].

Last but not least, the Freshness Theorem 2.1, not formalized in [12], was proved in the current development, with an equivalent handy version also proved/used in the Rocq specification:

$$(\forall a, b \in \mathbb{A}), ((\exists c \in \mathbb{A}), c \notin S_h \wedge c \# h(c)) \wedge a, b \# h \Rightarrow h(a) = h(b) \tag{11}$$

3 Rocq Development

In previous work [18], we proposed a typeclass hierarchy, with the bundle/unbundle approach [25], to formalize nominal sets in a constructive setting. It follows closely [12], a category theory oriented formalization in Agda. A particular difference in our development is the heavy employment of Rocq typeclass based generalized rewriting to mitigate the setoid burden reported in [12].

In this section, we present key aspects in the current approach. In particular, further developments since [18], such as the addition of the concepts of freshness, name abstraction, supported functions and related properties, *e.g.* the Freshness Theorem, not formalized in [12]. Remark that permutations are considered for a set of names $\mathbb{A}$. In some places, we use `(* ... *)` to indicate that part of the code was omitted for brevity, highlighting parts of code presented explicitly.

Names The set of atoms is defined as an abstract data type satisfying two properties: infiniteness and decidable equality of its elements. Defined using the Rocq module system[5], the set of natural numbers

[4] One would need to prove that `Prop` is a nominal set, demanding a nominal calculus of inductive constructions.

[5] Modules in Rocq/OCaML are similar to interfaces in Java.

is used to implement the module (Code 1).

```
Module Type ATOMIC.              Module Atom: ATOMIC.
  Parameter t : Set.               Definition t := nat.
  Axiom dec : EqDecision t.        Instance dec : EqDecision t := Nat.eq_dec.
  Axiom inf : Infinite t.          Instance inf : Infinite t := nat_infinite.
End ATOMIC.                      End Atom.

Notation Name := Atom.t.
Definition NameSet := (listset Name).
```

Code 1: Names

The infinite axiom for names is necessary to implement the choose-a-fresh-name principle (see Section 2). From a proof of infinite for a type `X`, *e.g.*, `NameSet` in Code 1, the coq-std++ library[6], an extended standard Rocq library developed by the same authors of the Iris project [16], makes available a lemma `exist_fresh: ∃ x, x ∉ X`.

Permutations We follow [12, 28] and specify name permutations as a list of swaps with operation `perm_swap` that applies the permutation to a name (Code 2). Every such a permutation in `Perm` is then finite and computation of composition and inverse are trivial, corresponding to list concatenation and reverse, respectively. For the sake of comparison, a representation based on an abstract notion of bijections would need an extra property, addressing finiteness, and the existence of inverses in the general case would need the axiom of choice.

```
Definition Swap: Type := (Name × Name).
Definition swap ('(a,b): Swap) (c: Name): Name :=
  if decide (a = c) then b else if decide (b = c) then a else c.
Notation "⟨ a , b ⟩" := (@cons Swap (a,b) nil).
Notation Perm := (list Swap).
Definition perm_swap (p: Perm) (a: Name): Name :=
  foldl (fun x y => swap y x) a p.
```

Code 2: Swap and Permutation.

Such a representation forms a group, with concatenation as the binary operator ($+$), empty list as the neutral element (ε) and inverse function as the list reverse function ($-$). Permutation representation is not unique though, *e.g.* $[(a,a)]$ and `nil` (empty list) both correspond to the identity function `id`. We consider them as a setoid instead [9], *i.e.* as the set of lists of swaps modulo an equivalence relation for permutations, corresponding to an instance of functional extensionality which is provable in the constructive setting. Besides defining an instance of the typeclass `Equivalence` –by providing an equivalence relation–, we need to provide instances of the `Proper` typeclass –proving that a function preserves the equivalence relations of its domain and codomain–, to enable the rewriting of equivalent terms when defining a setoid in Rocq. For instance, considering the binary operator of some group `A` and an equivalence relation over the same group, denoted by $\equiv$`@{A}`, `Proper` is defined as:

[6] https://plv.mpi-sws.org/coqdoc/stdpp/

```
grp_op_proper: Proper ((≡@{A}) ⇒ (≡@{A}) ⇒ (≡@{A})) (+)
```

which unfolds to a proof stating that if the arguments of the operator are equivalent, then so must be their results. We remark that, in some cases, Rocq could not find the correct relations to rewrite, so we opted to bundle equivalent and proper relations into their respective typeclasses (*e.g.* Code 3 below).

Nominal Sets Perm-set is defined as the typeclass `PermT`, with a permutation action over a set `X` (Code 3). The carrier set `X` is a setoid (`gact_setoid`) thus the permutation action needs to respect equivalences on both `Perm` and `X` sets, realised by the `Proper` typeclass (`gact_proper`).

```
Class PermAction (X: Type) := action: Perm → X → X.
Infix "•" := action.

Class PermT (X: Type) `{PermAction X, Equiv X}: Prop := {
  gact_setoid :> Equivalence(≡@{X});
  gact_proper :> Proper ((≡@{Perm}) ==> (≡@{X}) ==> (≡@{X})) (•);

  gact_id: ∀ (x: X), ε • x ≡@{X} x;
  gact_compat: ∀ (p q: Perm) (x: X), p • (q • x) ≡@{X} (q + p) • x
}.
```

Code 3: (Name) Permutation Action

The `Nominal` typeclass is a direct extension of `PermT`, since nominal sets are Perm-sets such that all members are finitely supported, as indicated by the `:>` operator defining the `nperm` field as a coercion from `Nominal` to `PermT` (Code 4). Moreover, the `support_spec` field corresponds to the property that any `x` in `X` is supported by `(support x)`, *i.e.* the `support` function maps each `x` to some support S_x, as characterised by (8).

```
Class Support X := support: X → NameSet.

Class Nominal (X: Type) `{Support X}: Prop := {
    nperm :> PermT X;
    support_spec: ∀ (x: X) (a b: Name),
        a ∉ support x → b ∉ support x → ⟨a,b⟩•x ≡@{X} x
}.
```

Code 4: Nominal Set

A protocol to define a nominal set N consists of five steps: (1) define a permutation action on N, (2) define a support function, (3) define and prove an equivalence relation for the elements of N, (4) prove an instance of `PermT` with N as the carrier set, (5) prove an instance of `Nominal` for N.

Freshness Constructive freshness is defined as in (9), where an equivalent definition, proved by the some/any principle `fresh_some_any_iff`, is also presented (see Code 5).

```
Definition freshP_e `{Nominal X} (a: Name) (x: X): Prop :=
   ∃ (b: Name), b ∉ support x ∧ ⟨a,b⟩ • x ≡@{X} x.

Definition freshP_a `{Nominal X} (a: Name) (x: X): Prop :=
   ∀ (b: Name), b ∉ support x → ⟨a,b⟩ • x ≡@{X} x.

Infix "#" := freshP_e. Infix "#ₐ" := freshP_a.

Lemma fresh_some_any_iff `{Nominal X} (a: Name) (x: X): a # x ↔ a #ₐ x.
```

Code 5: Constructive Freshness + Some/Any Freshness Lemma

Name Abstraction Our first implementation of the name abstraction set was a direct adaptation from Agda [12] into Rocq, *i.e.* as a pair of name and term, with the corresponding equivalence relation to represent $\approx_\alpha$ (see Section 2). The usage of name/term pairs turned out to be problematic while working with supported functions that also had another set of name/term pairs as domain. Our definition of supported functions are uncurried, thus n-ary functions have their domain represented by n-tuples. Consequently, the typeclass instance search algorithm assigned $\approx_\alpha$ for any function with such a domain, even when there is no equivalence relation defined for the set of pairs. A new Record type called `NameAbstraction`, used to represent name abstraction, solved the issue. In other words, instead of defining the equivalence relation over sets of the form $\mathbb{A} \times X$, X a nominal set, it is defined over this new type, avoiding the assignment of a $\approx_\alpha$ relation by the typeclass instance search algorithm for such a pair with no equivalence relation defined over the set. As a result, the nominal α-equivalence ($\approx_\alpha$) is defined as a relation over the `NameAbstraction` type.

```
Record NameAbstraction `{Nominal X} := mkAbstraction { name: Name; term: X }.
Notation " '[𝔸]' X " := (NameAbstraction X).
Notation " ⟦ a ⟧ x " := ({| name := a; term := x |}).

Instance alpha_equiv_e `{Nominal X}: Equiv [𝔸]X | 0 :=
   λ x y, ∃ (c: Name), c #[x.(name), y.(name), x.(term), y.(term)] ∧
   ⟨c,x.(name)⟩ • x.(term) ≡@{X} ⟨c,y.(name)⟩ • y.(term).

Instance alpha_equiv_a `{Nominal X}: Equiv [𝔸]X | 1 :=
   λ x y, ∀ (c: Name), c #[x.(name), y.(name), x.(term), y.(term)] →
   ⟨c,x.(name)⟩ • x.(term) ≡@{X} ⟨c,y.(name)⟩ • y.(term).

Infix "≈α" := (alpha_equiv_e). Infix "≈αₐ" := (alpha_equiv_a).

Lemma alpha_some_any `{Nominal X} (a b: Name) (x y: X):
   ⟦a⟧x ≈α ⟦b⟧y ↔ ⟦a⟧x ≈αₐ ⟦b⟧y.

Instance `{Equiv X}: Equiv [𝔸]X := (≈α).
Instance `{PermT X}: PermT [𝔸]X. Proof. (* ... *) Qed.
Instance `{Nominal X}: Nominal [𝔸]X. Proof. (* ... *) Qed.
```

```
Lemma nabs_action `{Nominal X} p a (x: X): p • ([[a]]x) = [[p • a]](p • x).

Lemma nabs_support `{Nominal X} a (x: X): support [[a]]x = support x \ {[a]}.
```

Code 6: Name Abstraction

Proving that name abstraction is a nominal set follows pretty much the general protocol mentioned above. It is worth noticing that, once a some/any principle for name abstraction is proved, there are two possible instances for the name abstraction equivalence relation. Unlike Haskell, that only allows one instance per class per type, Rocq typeclasses does not have this restriction. So, Rocq provides a mechanism to prioritize one instance over another during instance search. The notation `Typeclass | n` indicates a priority level, with `n` a natural number, with 0 indicating the highest priority. Both permutation action and support for name abstraction are presented as lemmas in Code 6, making it comparable with the definitions from Section 2 thus improving the readability.

Finitely supported functions The definition of the nominal set of supported functions poses particular challenges. Early in the development, we approached them with full unbundle, where the record `FunSupp` becomes a `Class` which is parametrized by all its members, showing an instance of the `Nominal` typeclass for each function separately. A major drawback in the approach was the impossibility to derive an instance for a composition of two nominal functions from their instances. And so, we opted out for a bundled alternative, similar to how bundled morphisms are defined in the Lean MathLib [7]. Supported functions are records parametrized by their domain and codomain. The advantage of this technique is that, we can bundle the support and the respective properties necessary to prove that it is nominal and that it respects the domain and codomain equivalences. On the other side, a lot of extra code is necessary to make it behave as regular Rocq functions, *i.e.* that can be applied, computed and composed with other functions defined in the system.

```
Context (X Y: Type) `{Nominal X, Nominal Y}.
Record FunSupp: Type := mkFunSupp {
  f_car :> X → Y;
  f_supp: NameSet; (* Function support *)
  f_proper: Proper ((≡@{X}) ⇒ (≡@{Y})) f_car;
  f_supp_spec: ∀ (a b: Name), a ∉ f_supp → b ∉ f_supp →
      ∀ (x: X), (⟨a,b⟩•(f_car (⟨a,b⟩•x))) ≡@{Y} f_car x
}.

Notation " A '→ₛ' B " := (FunSupp A B).
Notation "'λₛ' x .. y , t" := (* ... *).
Notation "'λₛ[' S ']' x .. y , t" := (* ... *).

Context `{Nominal X, Nominal Y}.
Instance: Equiv (X →ₛ Y) := fun f g => ∀ (x: X), f x ≡@{Y} g x.
Instance: PermAction (X →ₛ Y) :=
                       fun (p: Perm) (f: X →ₛ Y) => (λₛ (x: X), p • f(-p • x)).
Instance: Support (X →ₛ Y) := fun fs => f_supp fs.
```

```
Instance: PermT (X →s Y). Proof. (* ... *) Qed.
Instance: Nominal (X →s Y). Proof. (* ... *) Qed.
```

Code 7: Finitely Supported Functions

The definition of supported functions highlights the necessity of working with setoids in an axiom-free constructive formalization. Defining the sets of supported functions with syntactic equality as the equivalence relation leads the proof of `gact_id` from `PermT` to a proof of function extensionality, which cannot be proven in Rocq since it is not constructive. Even thought consistent with the Rocq type system, setoids are used instead, to preserve the constructive character of the formalization.

Freshness Theorem While the **choose-a-fresh-name principle** states that it is always possible to choose a fresh name (see Section 2), the Freshness Theorem states the conditions in which the choice of the fresh variable is irrelevant. This is a new (constructive) formalization, not present in [12].

```
Context `{Nominal X} (h: Name →s X) (Hp: ∃ a, a # h ∧ a # (h a)).

Definition freshF: X := h (fresh (support h)).

Theorem freshness_theorem: ∀ a, a # h → (h a) ≡ freshF.

Corollary freshness_theorem_inj: ∀ a b, a # h → b # h → h a ≡ h b.
```

Code 8: Freshness Theorem

Applicability of `freshness_theorem` in the formalization, as stated (see Code 8), proved cumbersome to be used and automated, since the theorem is mostly used in proofs to rewrite an argument of the function `h` for a new fresh variable. The corollary `freshness_theorem_inj`, which is, in fact, a proof of uniqueness for `freshF` corresponding to (11) in Section 2, showed preferable in the development, since it allowed easier rewrites by explicitly stating the names involved.

Freshness Condition for Binders The FCB property plays an important role in the derivation of the α-structural combinators, but here we want to highlight some aspects of the FCB formalization, in particular when dealing with supported functions. The notation λ_s⟦S⟧ `x, t` introduces a new supported function with support `S`, `x` a set of arguments and `t` the body of the function. For the properties `f_proper` and `f_supp_spec`, the notation introduces holes, *i.e.* proof obligations, which can be resolved with the help of the `refine` tactic.

```
Context `{Nominal X} `{Nominal Y}.
Context (f: (Name * X) →s Y) (fcb: (∃ a, a # f ∧ ∀ x, a # f (a,x))).

Definition _f: [𝔸]X →s Y.
refine (
   λs⟦support f⟧ (ax: [𝔸]X), (
      let h: Name →s X :=
        λs⟦support (ax.(name)) ∪ support (ax.(term)) ∪ support f⟧ c,
         (f (c, ⟨ax.(name),c⟩ • ax.(term)))
```

```
      in freshF h
   )
).
Proof. (* ... *) Defined.

Lemma fcb_support: support f = support _f.
Lemma fcb_prop (a: Name): a ∉ support f → ∀ x: X, f (a,x) ≡ _f ⟦a⟧x.
```

Code 9: Freshness Condition for Binders

4 Related and Future Work

Related Works Presenting the support of some function is not trivial and even in [28] least support is replaced by some support in some cases. Support cannot be defined from domain/codomain only, *i.e.* from the type, of some functions, *e.g.* choice functions. A heuristic is used in such cases [28], where the function support is considered to be the union of the support of all free symbols in the function definition. For instance, if $f(x) = g(c,x)$ then $supp(f) \subseteq S(g) \cup S(c)$ where $supp$ is the least support and S is some support, *i.e.* the heuristic is based on "some support as an upper bound" as characterised by (8).

Choudhury follows the categorical presentation of nominal sets [20] hence their categorical properties were also proved in [12]. We follow an algebraic approach to nominal sets but, apart from that, there is no conceptual difference between our work and his. In fact, several solutions adopted in the former, such as the use of setoids to keep constructiveness, were adopted in the present work. There are also some differences due to the "ergonomy" of each proof assistant. For instance, freshness is a computing artefact in [12] while ours is "logical". However, a dependent register with a witness in `Set` and an existential formula in `Prop` are, in some sense, the same. While the former is used in Agda, where there is no such a distinction as `Set` and `Prop` in Rocq, the latter is more suitable for the formalization in Rocq. The main advantage in using the Rocq Proof Assistant with a constructive approach is the use of typeclasses combined with the generalized rewriting mechanism, bearing the burden of working with setoids.

More recently, another formalization of nominal sets in Agda was proposed in [17], exploring different representations of finite permutations, instead of restricting it to the usual composition of transpositions, proving their equivalence and defining normalization of composition of transpositions.

Towards α-structural principles So far, we have developed all basic notions from the nominal theory: permutation types, which provide a uniform interface for handling variables and binders via name permutation; support, which characterizes the set of names that a certain term depends on; freshness, the complementary notion of support, gives us means to decide whether a term depends upon a name; name abstraction, an abstract notion of α-equivalence for nominal terms; and supported functions. We have all the tools to tackle syntax with binding variables inside the proof assistant. For instance, we can define the substitution of terms for the λ-calculus and prove properties such as the composition of substitutions. But still, the process demands extensive boilerplate encodings, not much different than working with de Bruijn indices or with locally nameless terms[7]. Ideally, we want to identify terms up to α-equivalence, as in a traditional pen-and-paper proof, in which we implicitly choose a term from the α-equivalence class satisfying the Barendregt Variable Condition (BVC): all bound names are pairwise distinct and their set

[7] For a comparison among the three approaches in the formalization of the higher-order π-calculus see [3].

is disjoint from the set of free names [8]. The α-structural principles enable this implicit reasoning [19]. Instead of defining a function or proving properties over the recursive/inductive structure of terms, we define/prove it over the terms modulo α-equivalence. There is only one full α-principle specification presented in [28], in the classical setting. Previous attempts in the constructive setting include an axiomatic approach [6] and a specification introduced in [15]. The former represents a proof-of-concept, where the axioms including the α-structural principles would be generated by an external tool from a syntactic and semantic description of the language with binders, while the latter have the principles relying on an extra assumption about the properties of α-equivalence class of terms, called α-compatibility (see also [13, 14]).

Currently, we only have a derivation of an α-recursion principle for the λ-calculus, since its codomain can be any nominal set, while an α-induction principle would have `Prop` as its codomain. For any inductively defined type X in the Rocq Calculus of (Co)Inductive Constructions, an associated "structural principle" operating over predicates `P: X → Type` is generated. Both recursion and induction principles are applications of this general structural principle to predicates with different codomains: `Set` for recursions, and `Prop` for inductions. And so, both principles consider the same term structure. The same reasoning/term structure could then be used to derive an α-structural induction principle, but first we would have to show that `Prop` is a nominal set, implicating that the set of the Rocq Core language terms would be a nominal set.

Instead, we are only developing a nominal theory for any inductively defined `X: Set`. It is worth noticing that when a user shows a new inductive type `X: Set` to be nominal, we have a "confined" signature to work over it. All constructors can be seen as uninterpreted functions over `X`, and can be proved nominal with an appropriate definition of support. The semantics assigned to its constructors will come from all user-defined functions over `X`, which can be shown to be Nominal if both domain and codomain are also nominal. As an ongoing work, we are investigating some alternatives for an induction principle, with the induction principle from [15, 14] posing the best alternative so far.

References

[1] Luca Aceto (2019): *Interview with Murdoch J. Gabbay and Andrew M. Pitts, 2019 Alonzo Church Award Recipients*. *Bull. EATCS* 128.

[2] Thorsten Altenkirch (2017): *From setoid hell to homotopy heaven? The role of extensionality in Type Theory*. In: *Proc. TYPES 2017*, pp. 12–13. Available at `https://types2017.elte.hu/proc.pdf`. Accessed on 16.05.2025.

[3] Guillaume Ambal, Sergueï Lenglet & Alan Schmitt (2021): *HOπ in Coq*. *J. Autom. Reason.* 65(1), pp. 75–124, doi:10.1007/S10817-020-09553-0.

[4] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, Daniele Nantes-Sobrinho & Ana Cristina Rocha Oliveira (2019): *A formalisation of nominal α-equivalence with A, C, and AC function symbols*. *Theor. Comput. Sci.* 781, pp. 3–23, doi:10.1016/J.TCS.2019.02.020.

[5] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, Gabriel Ferreira Silva & Daniele Nantes-Sobrinho (2021): *Formalising nominal C-unification generalised with protected variables*. *Math. Struct. Comput. Sci.* 31(3), pp. 286–311, doi:10.1017/S0960129521000050.

[6] Brian Aydemir, Aaron Bohannon & Stephanie Weirich (2006): *Nominal Reasoning Techniques in Coq: (Extended Abstract)*. In: *LFMTP@FLoC*, *ENTCS* 174, Elsevier, pp. 69–77, doi:10.1016/J.ENTCS.2007.01.028.

[7] Anne Baanen (2025): *Use and Abuse of Instance Parameters in the Lean Mathematical Library*. *J. Autom. Reason.* 69(1), p. 1, doi:10.1007/S10817-024-09712-7.

[8] Hendrik Pieter Barendregt (1985): *The lambda calculus - its syntax and semantics*. *Studies in logic and the foundations of mathematics* 103, North-Holland.

[9] Gilles Barthe, Venanzio Capretta & Olivier Pons (2003): *Setoids in type theory*. *J. Funct. Program.* 13(2), pp. 261–293, doi:10.1017/S0956796802004501.

[10] Jesper Bengtson & Joachim Parrow (2007): *Formalising the* pi*-Calculus Using Nominal Logic*. In: *FoSSaCS*, *Lecture Notes in Computer Science* 4423, Springer, pp. 63–77, doi:10.1007/978-3-540-71389-0_6.

[11] Arthur Charguéraud (2012): *The Locally Nameless Representation*. *J. Autom. Reason.* 49(3), pp. 363–408, doi:10.1007/S10817-011-9225-2.

[12] Pritam Choudhury (2015): *Constructive Representation of Nominal Sets in Agda*. Master's thesis, University of Cambridge.

[13] Ernesto Copello, Nora Szasz & Álvaro Tasistro (2017): *Machine-checked Proof of the Church-Rosser Theorem for the Lambda Calculus Using the Barendregt Variable Convention in Constructive Type Theory*. In: *LSFA*, *ENTCS* 338, Elsevier, pp. 79–95, doi:10.1016/J.ENTCS.2018.10.006.

[14] Ernesto Copello, Nora Szasz & Álvaro Tasistro (2021): *Formalization of metatheory of the Lambda Calculus in constructive type theory using the Barendregt variable convention*. *Math. Struct. Comput. Sci.* 31(3), pp. 341–360, doi:10.1017/S0960129521000335.

[15] Ernesto Copello, Alvaro Tasistro, Nora Szasz, Ana Bove & Maribel Fernández (2015): *Alpha-Structural Induction and Recursion for the Lambda Calculus in Constructive Type Theory*. In: *LSFA*, *ENTCS* 323, Elsevier, pp. 109–124, doi:10.1016/J.ENTCS.2016.06.008.

[16] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal & Derek Dreyer (2015): *Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning*. In: *POPL*, ACM, pp. 637–650, doi:10.1145/2676726.2676980.

[17] Miguel Pagano & José E. Solsona (2022): *Nominal Sets in Agda - A Fresh and Immature Mechanization*. In: *LSFA*, *EPTCS* 376, pp. 67–80, doi:10.4204/EPTCS.376.7.

[18] Fabrício S. Paranhos & Daniel Ventura (2022): *Towards a Formalization of Nominal Sets in Coq*. In: *CoqPL 2022*, pp. 1–3. Available at https://popl22.sigplan.org/details/CoqPL-2022-papers/4/Towards-a-Formalization-of-Nominal-Sets-in-Coq.

[19] Andrew M. Pitts (2006): *Alpha-structural recursion and induction*. *Journal of the ACM* 53(3), pp. 459–506, doi:10.1145/1147954.1147961.

[20] Andrew M. Pitts (2009): *Nominal Sets*. Cambridge University Press, doi:10.1017/CBO9781139084673.

[21] Andrew M. Pitts (2016): *Nominal techniques*. *ACM SIGLOG News* 3(1), pp. 57–72, doi:10.1145/2893582.2893594.

[22] Andrew M. Pitts (2023): *Locally Nameless Sets*. *Proc. ACM Program. Lang.* 7(POPL), pp. 488–514, doi:10.1145/3571210.

[23] Matthieu Sozeau (2009): *A New Look at Generalized Rewriting in Type Theory*. *J. Formaliz. Reason.* 2(1), pp. 41–62, doi:10.6092/ISSN.1972-5787/1574.

[24] Matthieu Sozeau & Nicolas Oury (2008): *First-Class Type Classes*. In: *TPHOLs*, *LNCS* 5170, Springer, pp. 278–293, doi:10.1007/978-3-540-71067-7_23.

[25] Bas Spitters & Eelis van der Weegen (2011): *Type classes for mathematics in type theory*. *Math. Struct. Comput. Sci.* 21(4), pp. 795–825, doi:10.1017/S0960129511000119.

[26] Andrew Swan (2016): *An algebraic weak factorisation system on 01-substitution sets: a constructive proof*. *J. Log. Anal.* 8.

[27] Andrew Swan (2017): *Some Brouwerian Counterexamples Regarding Nominal Sets in Constructive Set Theory*. *ArXiv* 1702.01556, doi:10.48550/arXiv.1702.01556.

[28] Christian Urban (2008): *Nominal Techniques in Isabelle/HOL*. *J. Autom. Reason.* 40(4), pp. 327–356, doi:10.1007/S10817-008-9097-2.

[29] Xinyi Wan & Qinxiang Cao (2023): *Formalization of Lambda Calculus with Explicit Names as a Nominal Reasoning Framework*. In: *SETTA*, *LNCS* 14464, Springer, pp. 262–278, doi:10.1007/978-981-99-8664-4_15.