Proceedings Paper:
Popescu, A. (2024) Nominal recursors as epi-recursors. In: Hicks, M., (ed.) Proceedings of the ACM on Programming Languages. 50th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2023), 15-21 Jan 2023, Boston, Massachusetts, United States. Association for Computing Machinery, New York, NY, United States. EISSN: 2475-1421.

https://doi.org/10.1145/3632857

Nominal Recursors as Epi-Recursors

ANDREI POPESCU, University of Sheffield, UK

We study nominal recursors from the literature on syntax with bindings and compare them with respect to expressiveness. The term "nominal" refers to the fact that these recursors operate on a syntax representation where the names of bound variables appear explicitly, as in nominal logic. We argue that nominal recursors can be viewed as *epi-recursors*, a concept that captures abstractly the distinction between the constructors on which one actually recurses, and other operators and properties that further underpin recursion. We develop an abstract framework for comparing epi-recursors and instantiate it to the existing nominal recursors, and also to several recursors obtained from them by cross-pollination. The resulted expressiveness hierarchies depend on how strictly we perform this comparison, and bring insight into the relative merits of different axiomatizations of syntax. We also apply our methodology to produce an expressiveness hierarchy of nominal *corecursors*, which are principles for defining functions targeting infinitary non-well-founded terms (which underlie λ-calculus semantics concepts such as Böhm trees). Our results are validated with the Isabelle/HOL theorem prover.

CCS Concepts: • **Theory of computation → Logic and verification**.

Additional Key Words and Phrases: nominal recursion and corecursion, nominal logic, epi-(co)recuror, syntax with bindings, formal reasoning, theorem proving

ACM Reference Format:
Andrei Popescu. 2024. Nominal Recursors as Epi-Recursors. *Proc. ACM Program. Lang.* 8, POPL, Article 15 (January 2024), 32 pages. https://doi.org/10.1145/3632857

1 INTRODUCTION

Syntax with bindings is pervasive in λ-calculi, logics and programming languages. Powerful mechanisms for performing definitions and reasoning involving bindings are important for formalizing the meta-theory of such systems [Abel et al. 2017; Aydemir et al. 2005; Felty et al. 2018]. Central among these mechanisms are *recursion principles* (*recursors* for short), allowing one to define functions by recursing over the syntax—e.g., for syntactic translations, semantic interpretations, and static analysis.

Much research has been dedicated to devising such mechanisms, within three main paradigms: nominal / nameful, nameless / De Bruijn, and higher-order abstract syntax (HOAS). Each of the three paradigms has pros and cons discussed at length in the literature (e.g., [Abel et al. 2017; Berghofer and Urban 2007; Blanchette et al. 2019; Felty and Momigliano 2012; Norrish and Vestergaard 2007]). A major selling point of the nominal paradigm, of which the most prominent representative is nominal logic [Aydemir et al. 2007; Gabbay and Pitts 1999; Urban and Tasson 2005], is that it employs a formal representation that is close to the one used in textbooks and informal descriptions, where on the one hand the names of bound variables are shown explicitly, and on the other hand their particular choice is irrelevant. Moreover, definitions and reasoning within this paradigm mimic informal practice, such as avoiding the capturing of bound variables by conveniently choosing their names in definition and proof contexts [Copello et al. 2018; Pitts 2006; Urban et al. 2007].

A delicate subject, where the nominal paradigm must walk a tightrope to achieve its goals, is the recursion principles. The specific challenge for recursion here is that terms with bindings, which

Author's address: Andrei Popescu, University of Sheffield, Sheffield, UK, a.popescu@sheffield.ac.uk.

ACM 2475-1421/2024/1-ART15
https://doi.org/10.1145/3632857

are equated modulo (i.e., quotiented to) α-equivalence (§2.1), do not form a free, hence standardly recursable datatype. To overcome this problem, various nominal recursors have been proposed and successfully deployed in formal developments (e.g., [Gabbay and Pitts 1999; Norrish 2004; Pitts 2006; Popescu and Gunter 2011; Urban and Berghofer 2006]). These recursors come in a variety of formats and flavors: they use different operators and have different features that enhance their cores (§2.2).

This paper contributes a general, systematic account of nominal recursors, highlighting their underlying principles and inter-connections. We ask two questions. First, *what is a nominal recursor?* In particular, what are the essential features that nominal recursors from the literature have in common (§3)? After an analysis of what the existing recursors aim to achieve and how they operate (§3.1) and the uniform rephrasing of their original presentations using signatures and models (§3.2), we synthesize the concept of an *epi-recursor* (§3.3). This concept captures abstractly their essential behavior, which can be summarized as follows: On top of the constructor infrastructure specific to standard recursion, these recursors take advantage of additional infrastructure employing non-constructor operators, to make the recursive definitions go through. And indeed, all the considered nominal recursors, and others obtained by cross-pollinating them, are particular cases of epi-recursors (§3.4).

Second, *what does it mean for a nominal recursor to be more expressive than another, and how do the existing recursors compare?* (§4). Apart from its theoretical interest, this question is of practical importance for designers and developers of formal reasoning frameworks. We answer it by introducing two relations for comparing the strength of epi-recursors, which differ in the amount of effort required in simulating one recursor by another. The first, stricter relation (§4.1) follows naturally from the definition of epi-recursors. The second, laxer relation (§4.3) is more elaborate, and was inspired by previous efforts to make a nominal recursor work on a brittle terrain where syntax meets semantics (§4.2). Instantiating the two relations to compare the nominal recursors yields two different hierarchies of strength. The comparisons reveal some interesting phenomena about the relative merits of considering various combinations of operations and axioms. Quite surprisingly given the wide variability of the underlying infrastructures, the laxer comparison yields an almost flat hierarchy, revealing that most of the recursors have the same strength—but still revealing that the symmetric operators (swapping and permutation) fare better than the asymmetric ones (renaming and substitution).

Analogous questions make sense when moving from the inductive to the coinductive world (§5). Here, we deal with infinitary non-well-founded λ-terms where we allow an infinite number of constructor applications (§5.1) and we study *corecursors*, which are principles for defining functions not from but *to* the set of infinitary terms. While our abstract notion of epi-corecursor (§5.2) is perfectly dual to that of epi-recursor, this is far from the case with the nominal corecursor versus recursor instances. However, there are elements of duality between these instances which we explore systematically, establishing a similar but different nominal corecursor expressiveness hierarchy (§5.3).

We have mechanized the discussed nominal (co)recursors and their comparison results in the Isabelle/HOL theorem prover [Nipkow et al. 2002] (§6). An extended technical report published on arXiv [Popescu 2023b] accompanies the paper, giving more details about our results and their proofs.

2 BACKGROUND

This section provides background on syntax with bindings (§2.1) and recalls several nominal recursors recursion from the literature (§2.2).

2.1 Terms with Bindings

We work with the paradigmatic syntax of lambda-calculus, but our results generalize to arbitrary binding syntaxes, as in [Pitts 2006; Urban and Kaliszyk 2012]. Let Var be a countably infinite set of variables, ranged over by x, y, z. The set Tr of λ-terms, ranged over by t, s, is defined by the grammar:

$$t \ ::= \ \mathsf{Vr}\ x \ \mid \ \mathsf{Ap}\ t_1\ t_2 \ \mid \ \mathsf{Lm}\ x\ t$$

with the proviso that terms are equated (identified) modulo α-equivalence (a.k.a. naming equivalence). Thus, for example, $\mathsf{Lm}\ x\ (\mathsf{Ap}\ (\mathsf{Vr}\ x)\ (\mathsf{Vr}\ x))$ and $\mathsf{Lm}\ y\ (\mathsf{Ap}\ (\mathsf{Vr}\ y)\ (\mathsf{Vr}\ y))$ are considered to be the same term. We will often omit writing the injection Vr of variables into terms.

In more detail, the above definition means the following: One first defines the set PTr of *preterms* (also called "raw terms") to be freely generated by the grammar $p ::= \mathsf{PVr}\ x \mid \mathsf{PAp}\ p_1\ p_2 \mid \mathsf{PLm}\ x\ p$. Then one defines α-equivalence $\equiv : \mathsf{PTr} \to \mathsf{PTr} \to \mathsf{Bool}$ inductively and defines Tr by quotienting: $\mathsf{Tr} = \mathsf{PTr}/{\equiv}$. Finally, one proves that the preterm constructors are compatible with $\equiv$, which allows to define the constructors on terms: $\mathsf{Vr} : \mathsf{Var} \to \mathsf{Tr}$, $\mathsf{Ap} : \mathsf{Tr} \to \mathsf{Tr} \to \mathsf{Tr}$ and $\mathsf{Lm} : \mathsf{Var} \to \mathsf{Tr} \to \mathsf{Tr}$.

Working with terms rather than preterms has well-known advantages, including the substitution operator being well-behaved. This is why most formal and informal developments prefer terms. For the rest of this paper, we will focus on terms and mostly forget about preterms—the latter will show up only occasionally, when we discuss certain intuitions.

Let Perm denote the set of finite permutations (bijections of finite support) on variables, $\{\sigma : \mathsf{Var} \to \mathsf{Var} \mid \{x \mid \sigma\ x \neq x\}\ \text{finite}\ \}$. We will consider generalizations of some common operations and relations on terms, namely:

- the constructors $\mathsf{Vr} : \mathsf{Var} \to \mathsf{Tr}$, $\mathsf{Ap} : \mathsf{Tr} \to \mathsf{Tr} \to \mathsf{Tr}$ and $\mathsf{Lm} : \mathsf{Var} \to \mathsf{Tr} \to \mathsf{Tr}$
- (capture-avoiding) substitution $\_[\_\ /\_] : \mathsf{Tr} \to \mathsf{Tr} \to \mathsf{Var} \to \mathsf{Tr}$; e.g., we have $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [\mathsf{Ap}\ x\ x\ /\ y]\ = \mathsf{Lm}\ x'\ (\mathsf{Ap}\ x'\ (\mathsf{Ap}\ x\ x))$ for some $x' \neq x$
- (capture-avoiding) renaming $\_[\_\ /\_] : \mathsf{Tr} \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Tr}$, the restriction of substitution to variables, i.e., it substitutes variables for variables rather than terms for variables; e.g., we have $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [x\ /\ y] = \mathsf{Lm}\ x'\ (\mathsf{Ap}\ x'\ x)$ for some $x' \neq x$
- swapping $\_[\_ \wedge \_] : \mathsf{Tr} \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Tr}$; e.g., we have $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [x \wedge y] = \mathsf{Lm}\ y\ (\mathsf{Ap}\ y\ x)$
- permutation $\_[\_] : \mathsf{Tr} \to \mathsf{Perm} \to \mathsf{Tr}$; e.g., we have $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ z\ y))\ [x \mapsto y, y \mapsto z, z \mapsto x] = \mathsf{Lm}\ y\ (\mathsf{Ap}\ x\ z)$
- free-variables $\mathsf{FV} : \mathsf{Tr} \to \mathcal{P}(\mathsf{Var})$ (the powerset of Var); e.g., we have $\mathsf{FV}(\mathsf{Lm}\ x\ (\mathsf{Ap}\ y\ x)) = \{y\}$ when $y \neq x$
- freshness $\_\#\_ : \mathsf{Var} \to \mathsf{Tr} \to \mathsf{Bool}$; e.g., we have $x \# \mathsf{Lm}\ x\ x$, and $\neg\ x \# \mathsf{Lm}\ y\ x$ when $x \neq y$

We let $x \leftrightarrow y$ be the permutation that takes x to y, y to x and everything else to itself. Note that permutation generalizes swapping, in that $t[x \wedge y] = t[x \leftrightarrow y]$. Also, note that free variables and freshness are of course two faces of the same coin: a variable x is fresh for a term t (i.e., $x \# t$) if and only if it is not free in t (i.e., $x \notin \mathsf{FV}\ t$).

We will not give definitions for the above operators, but count on the reader's familiarity with them. The definitions can be done in several equivalent ways—see, e.g., [Barendregt 1985; Pitts 2006].

2.2 Nominal Recursors

Next we look at nominal recursors in their "natural habitat", using concepts and terminology used by the authors who introduced them. Later on, in §3, we will recast them in a uniform format. For convenience, we refer to these recursors by the additional operators they are based on; e.g., the "perm/free", or "swap/fresh" recursor (not forgetting though that not only the chosen operators, but also the axioms imposed on them are responsible for a recursor's behavior).

2.2.1 The Perm/Free Recursor. This is the best known nominal recursor, originating in the context of nominal logic [Gabbay and Pitts 1999]. In the form we present here, which does not require any special logical foundation (e.g., axiomatic nominal set theory), it is due to Pitts [2006], who builds on previous work by Gabbay and Pitts [1999] and Urban and Berghofer [2006].

Some preparations are needed for describing this recursor. $(\mathsf{Perm}, \mathsf{id}, \circ)$ forms a group, where id is the identity permutation and $\circ$ is composition. A *pre-nominal set* is a set equipped with a Perm-action, i.e., a pair $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ where A is a set and $\_[\_]^{\mathcal{A}} : A \to \mathsf{Perm} \to A$ is an action of Perm on A, i.e., is idle for identity ($a[\mathsf{id}]^{\mathcal{A}} = a$ for all $a \in A$) and compositional ($a[\sigma \circ \tau]^{\mathcal{A}} = a[\tau]^{\mathcal{A}}[\sigma]^{\mathcal{A}}$).

Given a pre-nominal set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$, an $a \in A$ and a set $X \subseteq \mathsf{Var}$, we say that a *is supported by* X, or X *supports* a, if $a[x \leftrightarrow y]^{\mathcal{A}} = a$ holds for all $x, y \in \mathsf{Var} \setminus X$. An element $a \in A$ is called *finitely supported* if there exists a finite set X that supports a. A *nominal set* is a pre-nominal set where every element is finitely supported. If $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ is a nominal set and $a \in A$, then the smallest set that supports a can be shown to exist—it is denoted by $\mathsf{supp}^{\mathcal{A}}(a)$ and called the *support of* a. Given two pre-nominal sets $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ and $\mathcal{B} = (B, \_[\_]^{\mathcal{B}})$, the set $F = (A \to B)$ of functions from A to B forms a pre-nominal set $\mathcal{F} = (F, \_[\_]^{\mathcal{F}})$ by defining $f[\sigma]$ to be the function that sends each $a \in A$ to $f(a[\sigma^{-1}])[\sigma]$. The set of terms with their Perm-action, $(\mathsf{Tr}, \_[\_])$, forms a nominal set, where the support of a term t consists of its free variables.

The recursion theorem states that it is possible to define a function g from terms to any other set provided A is equipped with a nominal-set structure and additionally has some "term-like" operators matching the variable-injection, application and λ-abstraction operator, satisfying a specific condition. Concretely, it states that there exists a unique function g that commutes with these operators:

Thm 1. [Gabbay and Pitts 1999; Pitts 2006] Let $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ be a nominal set and let $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$ and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ be functions, all supported by a finite set X of variables and such that the following freshness condition for binders (FCB) holds: there exists $x \in \mathsf{Var}$ such that $x \notin X$ and $x \#^{\mathcal{A}} \mathsf{Lm}^{\mathcal{A}}\ x\ a$ for all $a \in A$.

Then there exists a unique $g : \mathsf{Tr} \to A$ supported by X such that the following hold:
(1) $g\ (\mathsf{Vr}\ x) = \mathsf{Vr}^{\mathcal{A}}\ x$ (2) $g\ (\mathsf{Ap}\ t_1\ t_2) = \mathsf{Ap}^{\mathcal{A}}(g\ t_1)\ (g\ t_2)$ (3) $g\ (\mathsf{Lm}\ x\ t) = \mathsf{Lm}^{\mathcal{A}}\ x\ (g\ t)$ if $x \notin X$

Note that the recursor features a parameter set of variables X, and requires the term-like operators to be supported by X; in exchange, it guarantees that the defined function g is also supported by X; moreover, the recursive clause for Lm is conditioned by the abstracted variable x being fresh for X. The rationale of this X-parametrization is the modelling of Barendregt's famous variable convention [Barendregt 1985][p.26]: "If [the terms] $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables." According to this, functions can be defined on terms while conveniently assuming that the λ-abstracted variables do not clash with other variables in the context of the definition—in the perm/free recursor, the set of these other variables is over-approximated by X.

2.2.2 The Swap/Free Recursor. The next recursor is due to Norrish [2004], who takes the free-variable operator as a primitive—whereas in nominal logic this operator, called support, is defined in terms of permutation. While this distinction is not important in the concrete case of terms, it does matter when one discusses abstract "term-like" structure on target domains. Another difference from the perm/free recursor is in taking swapping rather than permutation as primitive.

Norrish's recursor employs *swapping structures*, which are sets equipped with swapping- and free-variable-like operators, namely triples $\mathcal{A} = (A, \_[\_\wedge\_]^{\mathcal{A}}, \mathsf{FV}^{\mathcal{A}})$ where $\_[\_\wedge\_]^{\mathcal{A}} : A \to \mathsf{Var} \to \mathsf{Var} \to A$ and $\mathsf{FV}^{\mathcal{A}} : A \to \mathcal{P}(\mathsf{Var})$ such that the following hold for all $x, y, z \in \mathsf{Var}$ and $a \in A$:

(i) $a[x \wedge x]^{\mathcal{A}} = a$ (ii) $a[x \wedge y]^{\mathcal{A}}[x \wedge y]^{\mathcal{A}} = a$
(iii) $x, y \notin \mathsf{FV}^{\mathcal{A}}\ a$ implies $a[x \wedge y] = a$ (iv) $x \in \mathsf{FV}^{\mathcal{A}}(a[y \wedge z]^{\mathcal{A}})$ if and only if $x[y \wedge z] \in \mathsf{FV}^{\mathcal{A}} a$

The set of terms with their swapping and free-variable operations, $(\mathsf{Tr}, \_[\_\wedge\_], \mathsf{FV})$, form a swapping structure. The recursion theorem says that, given a suitable "term-like" infrastructure on a set A, which includes A being a swapping structure, and factors in a set of parameter variables X, there exists a unique function from terms to A that commutes with the term-like operators in a manner that obeys Barendregt's variables convention. And the function commutes with swapping and preserves the free variables, again in a Barendregt-convention observing manner. (Norrish also considers dynamic parameters, but Pitts [2006, Ex. 5.6] shows how to encode these using static parameters.)

Thm 2. [Norrish 2004] Let $\mathcal{A} = (A, \_[\_\wedge\_]^{\mathcal{A}}, \mathsf{FV}^{\mathcal{A}})$ be a swapping structure, $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : (\mathsf{Tr} \times A) \to (\mathsf{Tr} \times A) \to A$ and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to (\mathsf{Tr} \times A) \to A$ some functions, and X a finite set of variables such that the following hold: (1) $\mathsf{FV}^{\mathcal{A}}(\mathsf{Vr}^{\mathcal{A}} x) \subseteq \{x\} \cup X$
(2) If $\mathsf{FV}^{\mathcal{A}} a_1 \subseteq \mathsf{FV}\ t_1 \cup X$ and $\mathsf{FV}^{\mathcal{A}}\ a_2 \subseteq \mathsf{FV}\ t_2 \cup X$ then $\mathsf{FV}^{\mathcal{A}}(\mathsf{Ap}^{\mathcal{A}}\ (t_1, a_1)\ (t_2, a_2)) \subseteq \mathsf{FV}\ (\mathsf{Ap}\ t_1\ t_2) \cup X$
(3) If $\mathsf{FV}^{\mathcal{A}}\ a \subseteq \mathsf{FV}\ t \cup X$ then $\mathsf{FV}^{\mathcal{A}}(\mathsf{Lm}^{\mathcal{A}}\ x\ (t, a)) \subseteq \mathsf{FV}\ (\mathsf{Lm}\ x\ t) \cup X$
(4) If $x, y \notin X$, then $(\mathsf{Vr}^{\mathcal{A}}\ z)\ [x \wedge y]^{\mathcal{A}} = \mathsf{Vr}^{\mathcal{A}}\ (z[x \wedge y])$
(5) If $x, y \notin X$, then $(\mathsf{Ap}^{\mathcal{A}}(t_1, a_1)\ (t_2, a_2))\ [x \wedge y]^{\mathcal{A}} = \mathsf{Ap}^{\mathcal{A}}(t_1[x \wedge y], a_1[x \wedge y]^{\mathcal{A}})\ (t_2[x \wedge y], a_2[x \wedge y]^{\mathcal{A}})$
(6) If $x, y \notin X$, then $(\mathsf{Lm}^{\mathcal{A}}\ z\ (t, a))\ [x \wedge y]^{\mathcal{A}} = \mathsf{Lm}^{\mathcal{A}}\ (z[x \wedge y])\ (t[x \wedge y], a[x \wedge y]^{\mathcal{A}})$
Then there exists a unique function $g : \mathsf{Tr} \to A$ such that the following hold:
(i) $g\ (\mathsf{Vr}\ x) = \mathsf{Vr}^{\mathcal{A}}\ x$ (ii) $g\ (\mathsf{Ap}\ t_1\ t_2) = \mathsf{Ap}^{\mathcal{A}}\ (t_1, g\ t_1)\ (t_2, g\ t_2)$
(iii) $g\ (\mathsf{Lm}\ x\ t) = \mathsf{Lm}^{\mathcal{A}}\ x\ (t, g\ t)$ if $x \notin X$ (iv) $g\ (t[x \wedge y]) = (g\ t)[x \wedge y]^{\mathcal{A}}$ if $x, y \notin X$
(v) $\mathsf{FV}^{\mathcal{A}}(g\ t) \subseteq \mathsf{FV}\ t \cup X$

An enhancement present in this recursor is the enabling of full-fledged (primitive) recursion rather than mere iteration—as seen in the constructor-like operators $\mathsf{Vr}^{\mathcal{A}}$, $\mathsf{Ap}^{\mathcal{A}}$ and $\mathsf{Lm}^{\mathcal{A}}$ taking as inputs not only elements of A but also terms. Hence the recursive clauses for g allow the computed value to depend not only on the recursive results for smaller terms, but also on the smaller terms themselves.

2.2.3 The Swap/Fresh Recursor. The next recursor was described by Gheri and Popescu [2020]. Similarly to the previous recursors, it uses structures that generalize term operators, here freshness and swapping. It is similar to the swap/free recursor by its focus on swapping, but different in that it (a) uses freshness rather than free variables, (b) requires different properties from the models, (c) does not support Barendregt's convention and (d) extends full-fledged recursion to non-constructor operators (in that these operators also take additional term arguments).

A *freshness-swapping model* is a set equipped with constructor-, swapping- and freshness-like operators, namely a tuple $\mathcal{A} = (A, \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}}, \_[\_\wedge\_]^{\mathcal{A}}, \#^{\mathcal{A}})$ where $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : (\mathsf{Tr} \times A) \to (\mathsf{Tr} \times A) \to A$, $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to (\mathsf{Tr} \times A) \to A$, $\_[\_\wedge\_]^{\mathcal{A}} : (\mathsf{Tr} \times A) \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Tr}$ and $\#^{\mathcal{A}} : \mathsf{Var} \to (\mathsf{Tr} \times A) \to \mathsf{Bool}$ satisfying: (1) $x \neq y$ implies $x\ \#^{\mathcal{A}}\ \mathsf{Vr}^{\mathcal{A}} y$
(2) $x \# t_1$, $x\ \#^{\mathcal{A}}\ (t_1, a_1)$, $x \# t_2$ and $x\ \#^{\mathcal{A}}\ (t_2, a_2)$ implies $x\ \#^{\mathcal{A}}\ \mathsf{Ap}^{\mathcal{A}}\ (t_1, a_1)\ (t_2, a_2)$
(3) $y = x$ or $[y \# t$ and $y\ \#^{\mathcal{A}}\ (t, a)]$ implies $y\ \#^{\mathcal{A}}\ \mathsf{Lm}^{\mathcal{A}}\ x\ (t, a)$
(4) $(\mathsf{Vr}\ x, \mathsf{Vr}^{\mathcal{A}}\ x)[y \wedge z]^{\mathcal{A}} = \mathsf{Vr}^{\mathcal{A}}(x[y \wedge z])$
(5) $(\mathsf{Ap}\ t_1\ t_2, \mathsf{Ap}^{\mathcal{A}}(t_1, a_1)\ (t_2, a_2))[y \wedge z]^{\mathcal{A}} = \mathsf{Ap}^{\mathcal{A}}(t_1[y \wedge z], (t_1, a_1)[y \wedge z]^{\mathcal{A}})\ (t_2[y \wedge z], (t_2, a_2)[y \wedge z]^{\mathcal{A}})$
(6) $(\mathsf{Lm}\ x\ t,\ \mathsf{Lm}^{\mathcal{A}}\ x\ (t, a))[y \wedge z]^{\mathcal{A}} = \mathsf{Lm}^{\mathcal{A}}\ (x[y \wedge z])\ (t[y \wedge z], (t, a)[y \wedge z]^{\mathcal{A}})$
(7) $z \notin \{x_1, x_2\}$, $z\ \#^{\mathcal{A}}\ (t_1, a_1)$, $z\ \#^{\mathcal{A}}\ (t_2, a_2)$ and $(t_1, a_1)[z \wedge x_1] = (t_2, a_2)[z \wedge x_2]$ implies $\mathsf{Lm}^{\mathcal{A}}\ x_1\ (t_1, a_1) = \mathsf{Lm}^{\mathcal{A}}\ x_2\ (t_2, a_2)$

The recursion theorem states that terms are the initial freshness-swapping model (hence initial in a certain Horn theory), i.e., for any freshness-swapping model there exists a unique function from terms that commutes with the constructors and swapping, and preserves freshness.

Thm 3. [Gheri and Popescu 2020] For any freshness-swapping model $\mathcal{A} = (A, \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}}, \_[\_\wedge\_]^{\mathcal{A}}, \#^{\mathcal{A}})$, there exists a unique function $g : \mathsf{Tr} \to A$ such that the following hold:

(i) $g\ (\mathsf{Vr}\ x) = \mathsf{Vr}^{\mathcal{A}}\ x$ (ii) $g\ (\mathsf{Ap}\ t_1\ t_2) = \mathsf{Ap}^{\mathcal{A}}\ (t_1, g\ t_1)\ (t_2, g\ t_2)$

(iii) $g\ (\mathsf{Lm}\ x\ t) = \mathsf{Lm}^{\mathcal{A}}\ x\ (t, g\ t)$ (iv) $g\ (t[x \wedge y]) = (t, g\ t)[x \wedge y]^{\mathcal{A}}$

(v) $x \#\ t$ implies $x \#^{\mathcal{A}}\ (t, g\ t)$

2.2.4 The Subst/Fresh Recursor. The next recursor, introduced by Popescu and Gunter [2011], has a similar structure to the previous one but uses substitution rather than swapping.

A *freshness-substitution model* is similar to a freshness-swapping model, but instead of a swapping-like operator it has a substitution-like operator $\_[\_/\_]^{\mathcal{A}} : (\mathsf{Tr} \times A) \to (\mathsf{Tr} \times A) \to \mathsf{Var} \to \mathsf{Tr}$ and:

- instead of clauses (4)–(6) of swapping commuting with the constructors, it satisfies similar clauses for substitution—but where commutation with λ-abstraction is restricted by a freshness condition
- instead of clause (7), it satisfies a substitution-based renaming clause for λ-abstraction.

Namely, it satisfies the following clauses: (4) $(\mathsf{Vr}\ x, \mathsf{Vr}^{\mathcal{A}}\ x)[(t, a)/z]^{\mathcal{A}} = (\text{if } x = z \text{ then } a \text{ else } \mathsf{Vr}^{\mathcal{A}} x)$

(5) $(\mathsf{Ap}\ t_1\ t_2,\ \mathsf{Ap}^{\mathcal{A}}\ (t_1, a_1)\ (t_2, a_2))[(s, b)/z]^{\mathcal{A}} =$
$\mathsf{Ap}^{\mathcal{A}}(t_1[s/z], (t_1, a_1)[(s, b)/z]^{\mathcal{A}})\ (t_2[s/z], (t_2, a_2)[(s, b)/z]^{\mathcal{A}})$

(6) $x \neq z$ and $x \#^{\mathcal{A}} (s, b)$ implies
$(\mathsf{Lm}\ x\ t, \mathsf{Lm}^{\mathcal{A}} x\ (t, a))[(s, b)/z]^{\mathcal{A}} = \mathsf{Lm}^{\mathcal{A}} x\ (t[s/z], (t, a)[(s, b)/z]^{\mathcal{A}})$

(7) $z \neq x$ and $z \#^{\mathcal{A}}\ (t, a)$ implies $\mathsf{Lm}^{\mathcal{A}}\ z\ [(t, a)[(\mathsf{Vr}\ z, \mathsf{Vr}^{\mathcal{A}}\ z)/x]] = \mathsf{Lm}^{\mathcal{A}}\ x\ (t, a)$

Thm 4. [Popescu and Gunter 2011] For any freshness-substitution model $\mathcal{A} = (A, \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}}, \_[\_/\_]^{\mathcal{A}}, \#^{\mathcal{A}})$, there exists a unique $g : \mathsf{Tr} \to A$ such that the clauses listed in Thm. 3 hold, except that the clause for swapping is replaced by a clause for substitution: $g\ (t[s/y]) = (t, g\ t)[(s, g\ s)/y]^{\mathcal{A}}$.

2.2.5 The Renaming Recursor. The next recursor was introduced by Popescu [2023c]. Besides the constructors, it only uses one operator, renaming—subject to an equational theory described next.

A *constructor-enriched renset* is a tuple $\mathcal{A} = (A, \_[\_/\_]^{\mathcal{A}}, \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ where $\_[\_/\_]^{\mathcal{A}} : A \to \mathsf{Var} \to \mathsf{Var} \to A$, $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$ and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ are such that the following hold: (1) $a[x/x]^{\mathcal{A}} = a$ (2) If $x_1 \neq y$ then $a[x_1/y]^{\mathcal{A}}[x_2/y]^{\mathcal{A}} = a[x_1/y]$

(3) $y \neq x_2$ then $a[y/x_2]^{\mathcal{A}}[x_2/x_1]^{\mathcal{A}}[x_3/x_2]^{\mathcal{A}} = a[y/x_2]^{\mathcal{A}}[x_3/x_1]^{\mathcal{A}}$

(4) If $x_2 \neq y_1 \neq x_1 \neq y_2$ then $a[x_2/x_1]^{\mathcal{A}}[y_2/y_1]^{\mathcal{A}} = a[y_2/y_1]^{\mathcal{A}}[x_2/x_1]^{\mathcal{A}}$

(5) $(\mathsf{Vr}^{\mathcal{A}}\ x)[y/z]^{\mathcal{A}} = \mathsf{Vr}^{\mathcal{A}}(x[y/z])$ (6) $(\mathsf{Ap}^{\mathcal{A}}\ a_1\ a_2)[y/z]^{\mathcal{A}} = \mathsf{Ap}^{\mathcal{A}}(a_1[y/z]^{\mathcal{A}})\ (a_2[y/z]^{\mathcal{A}})$

(7) if $x \notin \{y, z\}$ then $(\mathsf{Lm}^{\mathcal{A}}\ x\ a)[y/z]^{\mathcal{A}} = \mathsf{Lm}^{\mathcal{A}}\ x\ (a[y/z]^{\mathcal{A}})$ (8) $(\mathsf{Lm}^{\mathcal{A}}\ x\ a)[y/x]^{\mathcal{A}} = \mathsf{Lm}^{\mathcal{A}}\ x\ a$

(9) if $z \neq y$ then $\mathsf{Lm}^{\mathcal{A}}\ x\ (a[z/y]^{\mathcal{A}}) = \mathsf{Lm}^{\mathcal{A}}\ y\ (a[z/y]^{\mathcal{A}}[y/x]^{\mathcal{A}})$

Equations (1)–(3) refer to standard properties of renaming, while (4)–(9) connect renaming and the constructors. The recursion theorem characterizes terms as initial model in this equational theory.

Thm 5. [Popescu 2023c] For any constructor-enriched renamable set $\mathcal{A} = (A, \_[\_/\_]^{\mathcal{A}}, \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$, there exists a unique $g : \mathsf{Tr} \to A$ such that the following hold:

(i) $g\ (\mathsf{Vr}\ x) = \mathsf{Vr}^{\mathcal{A}}\ x$ (ii) $g\ (\mathsf{Ap}\ t_1\ t_2) = \mathsf{Ap}^{\mathcal{A}}\ (g\ t_1)\ (g\ t_2)$

(ii) $g\ (\mathsf{Lm}\ x\ t) = \mathsf{Lm}^{\mathcal{A}}\ x\ (g\ t)$ (iv) $g\ (t[x/y]) = (g\ t)[x/y]^{\mathcal{A}}$

2.2.6 Enhancements. The above recursors clearly have many aspects in common, but also display some essential variability regarding the non-constructor operators they are based on and the conditions imposed on the target-domain counterparts of these operators. Other dimensions of variability were what we called the "enhancements": support for Barendregt's convention and full-fledged recursion. It turns out that both types of enhancements can be made uniformly to all nominal recursors (as we detail in Popescu [2023b, App. D]). So in what follows, for comparing these recursors we will strip them of their enhancements and focus on their essential variability only.

3 NOMINAL RECURSORS AS EPI-RECURSORS

In this section, we will propose regarding nominal recursors as mechanisms for helping recursion to proceed "as if freely", i.e., by writing clauses for each constructor as if the datatype of terms were freely generated by the constructors. We start by describing this view informally on an example (§3.1). To formalize the view, we introduce signatures and models that describe uniformly the term-like operators featured in the previous section's recursion theorems (§3.2). Then we define the central concept of this paper, that of an epi-recursor (§3.3), which captures this view in a general category-theoretic form. Finally, we show that all the discussed nominal recursors, and others that are obtained as variations or combinations of these, are epi-recursors (§3.4).

As mentioned, we will not consider the recursors in their original forms—as introduced by their authors, recalled in §2.2—but their essential cores, stripped of their full-fledged recursion and Barendregt convention enhancements. (The enhancements, discussed in Popescu [2023b, App. D], turn out to be orthogonal.)

3.1 The Purpose of Nominal Recursors

Let us start with recursion over a free datatype, i.e., freely generated by the constructors, such as that of preterms (recalled in §2.1). To define a function $g : \mathsf{PTr} \to A$ between preterms and some target domain A, informally speaking we write recursive clauses for each of the constructors:

- $g\ (\mathsf{PVr}\ x)\ =\ \langle$expression depending on $x\rangle$
- $g\ (\mathsf{PAp}\ p_1\ p_2)\ =\ \langle$expression depending on $g\ p_1$ and $g\ p_2\rangle$
- $g\ (\mathsf{PLm}\ x\ p)\ =\ \langle$expression depending on x and $g\ p\rangle$

The above "expression depending on" formulation can be made rigorous by considering preterm-like operations on the target domain A. Namely, for a recursive definition like the above to be possible, we must organize A as a model $\mathcal{A} = (A, \mathsf{PVr}^{\mathcal{A}}, \mathsf{PAp}^{\mathcal{A}}, \mathsf{PLm}^{\mathcal{A}})$, where $\mathsf{PVr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{PAp}^{\mathcal{A}} : A \to A \to A$ and $\mathsf{PLm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$. Now, the recursive definition of g is nothing but the statement that g commutes with the operations that correspond to each other:

$$g\ (\mathsf{PVr}\ x) = \mathsf{PVr}^{\mathcal{A}}\ x \qquad g\ (\mathsf{PAp}\ p_1\ p_2) = \mathsf{PAp}^{\mathcal{A}}\ (g\ p_1)\ (g\ p_2) \qquad g\ (\mathsf{PLm}\ x\ p) = \mathsf{PLm}^{\mathcal{A}}\ x\ (g\ p)$$

In fact, we could say that the model $\mathcal{A}$ *is* the recursive definition of g—because it determines a unique function $g : \mathsf{PTr} \to A$ that commutes with the operations.

Now, let's switch from preterms to terms. We can summarize the purpose of all nominal recursors:

> to define functions $g : \mathsf{Tr} \to A$ between terms and target domains A by recursing over the constructors *as if the datatype of terms was freely generated,*

i.e., by writing recursive clauses similarly to those of the free datatype of preterms:

- $g\ (\mathsf{Vr}\ x)\ =\ \langle$expression depending on $x\rangle$
- $g\ (\mathsf{Ap}\ t_1\ t_2)\ =\ \langle$expression depending on $g\ t_1$ and $g\ t_2\rangle$
- $g\ (\mathsf{Lm}\ x\ t)\ =\ \langle$expression depending on x and $g\ t\rangle$

But the datatype of terms is not freely generated, so such a definition cannot work out of the box. One needs to further underpin recursion by describing the interaction of the intended function g not only with the constructors, but also with other operators. For example, the swap/fresh recursor described in §2.2.3 requires two additional clauses, for the swapping and freshness operators:

- $g\ (t[x \wedge y])\ =\ \langle$expression depending on x, y and $g\ t\rangle$
- $x\ \#\ t$ implies $\langle$expression depending on x and $g\ t\rangle$

This is also made rigorous using models. The requirement is to define term-like operators on the target domain A corresponding not only to the constructors but also to other operators; i.e., in this case, organize A as a model $\mathcal{A} = (A, \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}}, \_[\_ \wedge \_]^{\mathcal{A}}, \#^{\mathcal{A}})$, consisting of:

- (as before for preterms) counterparts of the constructors, $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$, and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$,
- as well as counterparts of the swapping operation and the freshness relation, $\_[\_ \wedge \_]^{\mathcal{A}} : A \to \mathsf{Var} \to \mathsf{Var} \to A$ and $\#^{\mathcal{A}} : \mathsf{Var} \to A \to \mathsf{Bool}$

Another new requirement compared to the case of free datatypes is that the model $\mathcal{A}$ is similar to terms not only in the matching arities of its operators, but also in satisfying specific term-like properties, i.e, $\mathcal{A}$-counterparts of properties of the terms—e.g., swapping commuting with λ-abstraction.

If the above is successfully achieved, i.e., if one provides a model $\mathcal{A}$ satisfying the required properties, then the recursor guarantees the existence of a unique function $g : \mathsf{Tr} \to A$ commuting with the operations (here, constructors and swapping) and preserving the relations (here, freshness).

The next simple example illustrates the above discussion. §4.2 and Popescu [2023b, App. B] show more examples; many others can be found in, e.g., [Norrish 2004; Pitts 2006; Popescu and Gunter 2011].

Example 6. *(number of free occurrences)* Let us consider the task of defining the function $\mathsf{noccs} : \mathsf{Tr} \to (\mathsf{Var} \to \mathbb{N})$, where $\mathsf{noccs}\ t\ x$ counts the number of (free) occurrences of the variable x in the term t. The natural recursive clauses we would wish to write are

(i) $\mathsf{noccs}\ (\mathsf{Vr}\ y)\ x = (\text{if } x = y \text{ then } 1 \text{ else } 0)$ (ii) $\mathsf{noccs}\ (\mathsf{Ap}\ t_1\ t_2)\ x = \mathsf{noccs}\ t_1\ x + \mathsf{noccs}\ t_2\ x$

(iii) $\mathsf{noccs}\ (\mathsf{Lm}\ y\ t)\ x = (\text{if } x = y \text{ then } 0 \text{ else } \mathsf{noccs}\ t\ x)$

As discussed, such a definition does not work out of the box (in that, in itself, it does not constitute a correct recursive definition) because of the non-freeness of the terms. To make this work, we can add clauses describing the intended behavior of noccs with respect to swapping and freshness:

(iv) $\mathsf{noccs}\ (t[y_1 \wedge y_2])\ x = \mathsf{noccs}\ t\ (x[y_1 \wedge y_2])$ (v) $x \#\ t$ implies $\mathsf{noccs}\ t\ x = 0$

This means organizing the target domain $\mathsf{Var} \to \mathbb{N}$ as a model $\mathcal{A}$ by defining the following operators:

- $\mathsf{Vr}^{\mathcal{A}} = (\lambda x.\ \text{if } x = y \text{ then } 1 \text{ else } 0)$
- $\mathsf{Ap}^{\mathcal{A}}\ m_1\ m_2 = (\lambda x.\ m_1\ x + m_2\ x)$
- $\mathsf{Lm}^{\mathcal{A}}\ y\ m = (\lambda x.\ \text{if } x = y \text{ then } 0 \text{ else } m\ x)$
- $m\ [y_1 \wedge y_2]^{\mathcal{A}} = (\lambda x.\ m\ (x[y_1 \wedge y_2]))$
- $x\ \#^{\mathcal{A}}\ m = (m\ x = 0)$

After checking that $\mathcal{A}$ satisfies some required properties (which in this case are trivial arithmetic properties) we obtain a unique function noccs satisfying clauses (i)–(v).

3.2 Signatures and Models

Next we introduce notation that allows us to discuss the various recursors uniformly. Let Sym, the *set of (operation or relation) symbols*, be {vr, ap, lm, pm, sw, sb, ren, fv, fr}. The symbols refer to variable, application and λ-abstraction constructors, permutation, swapping, substitution, renaming and free-variable operations, and the freshness relation, respectively. A *signature* Σ will be any subset of Sym.

Given a signature Σ, a Σ-*model* $\mathcal{M}$ consists of a set M, called the *carrier set*, and operations and/or relations on M as indicated in the signature. More precisely: if $\mathsf{vr} \in \Sigma$ then $\mathcal{M}$ has an operation $\mathsf{Vr}^{\mathcal{M}} : \mathsf{Var} \to M$; if $\mathsf{ap} \in \Sigma$ then $\mathcal{M}$ has an operation $\mathsf{Ap}^{\mathcal{M}} : M \to M \to M$; if $\mathsf{lm} \in \Sigma$ then $\mathcal{M}$ has $\mathsf{Lm}^{\mathcal{M}} : \mathsf{Var} \to M \to M$; if $\mathsf{pm} \in \Sigma$ then $\mathcal{M}$ has $\_[\_]^{\mathcal{M}} : M \to \mathsf{Perm} \to M$; if $\mathsf{sw} \in \Sigma$ then $\mathcal{M}$ has $\_[\_ \wedge \_]^{\mathcal{M}} : M \to \mathsf{Var} \to \mathsf{Var} \to M$; if $\mathsf{sb} \in \Sigma$ then $\mathcal{M}$ has $\_[\_\ /\_]^{\mathcal{M}} : M \to M \to \mathsf{Var} \to M$; if $\mathsf{ren} \in \Sigma$ then $\mathcal{M}$ has $\_[\_\ /\_]^{\mathcal{M}} : M \to \mathsf{Var} \to \mathsf{Var} \to M$; if $\mathsf{fv} \in \Sigma$ then $\mathcal{M}$ has $\mathsf{FV}^{\mathcal{M}} : M \to \mathcal{P}(\mathsf{Var})$; if $\mathsf{fr} \in \Sigma$ then $\mathcal{M}$ has $\#^{\mathcal{M}} : \mathsf{Var} \to M \to \mathsf{Bool}$.

Given two Σ-models $\mathcal{M}$ and $\mathcal{M}'$, a *morphism* between them is a function between their carrier sets $g : M \to M'$ that commutes with the operations and preserves the relations. For example: if $\mathsf{vr} \in \Sigma$, we require that $g(\mathsf{Vr}^{\mathcal{M}}\ x) = \mathsf{Vr}^{\mathcal{M}'} x$; if $\mathsf{lm} \in \Sigma$, we require that $g(\mathsf{Lm}^{\mathcal{M}}\ x\ m) = \mathsf{Lm}^{\mathcal{M}'} x\ (g\ m)$; if $\mathsf{fr} \in \Sigma$, we require that $x\ \#^{\mathcal{M}}\ m$ implies $x\ \#^{\mathcal{M}'}\ (g\ m)$; if $\mathsf{fv} \in \Sigma$, we require that $\mathsf{FV}^{\mathcal{M}'}(g\ m) \subseteq \mathsf{FV}^{\mathcal{M}}\ m$. We write $g : \mathcal{M} \to \mathcal{M}'$ to indicate that the function g is a morphism between $\mathcal{M}$ and $\mathcal{M}'$. Σ-models

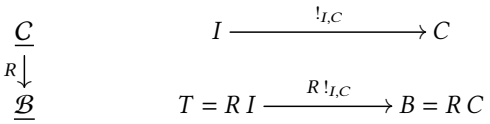

Fig. 1. Epi-recursor in action

and their morphisms form a category. We write $\mathcal{T}r(\Sigma)$ for the Σ-model whose carrier is the set of terms Tr and whose operations and relations are the standard ones for terms.

Let $\Sigma_{\mathsf{ctor}} = \{\mathsf{vr}, \mathsf{lm}, \mathsf{ap}\}$ be the signature comprising the constructor symbols only. Ignoring the full-fledged recursion and Barendregt enhancements, what all the described nominal recursors have in common, which is also shared with the standard recursors over free datatypes, is that they allow one to recurse over terms using constructors, i.e., they (1) require the intended target domain to be (at least) a Σ_{ctor}-model $\mathcal{M}$, and (2) ensure the existence of a function g that commutes with the constructors, i.e., a morphism $g : \mathcal{T}r(\Sigma_{\mathsf{ctor}}) \to \mathcal{M}$. Also, as illustrated in §3.1, another aspect that the nominal recursors have in common is that, to make recursing over terms possible, they (1) require extending $\mathcal{M}$ to a Σ_{ext}-model $\mathcal{M}'$ for an extended signature $\Sigma_{\mathsf{ext}} \supseteq \Sigma_{\mathsf{ctor}}$ and verifying certain properties for $\mathcal{M}'$, and (2) capitalize on the fact that $\mathcal{T}r(\Sigma_{\mathsf{ext}})$ is initial among Σ_{ext}-models that satisfy these properties—which yields a morphism $\mathcal{T}r(\Sigma_{\mathsf{ext}}) \to \mathcal{M}'$, i.e., a function g that commutes not only with the constructors but also with the other operators in Σ_{ext}. In short, what all these recursors do is *underpin constructor-based recursion by extending the signature and exploiting initiality of the term model there.*

3.3 Epi-Recursors

We capture the above phenomenon in the following concept:

Def 7. An *epi-recursor* is a tuple $r = (\underline{\mathcal{B}}, T, \underline{\mathcal{C}}, I, R)$ where:

- $\underline{\mathcal{B}}$ is a category called *the base category*
- $\underline{\mathcal{C}}$ is a category called *the extended category*
- $R : \underline{\mathcal{C}} \to \underline{\mathcal{B}}$ is a functor such that $R\,I = T$
- T is an object in $\underline{\mathcal{B}}$ called *the base object*
- I is an initial object in $\underline{\mathcal{C}}$

In typical examples $\underline{\mathcal{C}}$ and $\underline{\mathcal{B}}$ will be categories of models, i.e., sets with algebraic/relational structure, so that the models in $\underline{\mathcal{C}}$ have more structure than those in $\underline{\mathcal{B}}$, and R will be a structure-forgetting functor. The base object T will be the syntactic model of interest—such as the term model $\mathcal{T}r(\Sigma_{\mathsf{ctor}})$ with constructors only—which is the source object of the intended recursive definitions. Then I is its extension to an object of $\underline{\mathcal{C}}$ that makes recursion possible—for our nominal recursors, this is a model $\mathcal{T}r(\Sigma_{\mathsf{ext}})$, having other "recursion-underpinning" operators besides the constructors.

To define a morphism $g : T \to B$ in $\underline{\mathcal{B}}$ (to some object B in $\underline{\mathcal{B}}$) using the epi-recursor $r = (\underline{\mathcal{B}}, T, \underline{\mathcal{C}}, I, R)$, we do the following (see Fig. 1): (1) extend B to an object C in $\underline{\mathcal{C}}$ (with $R\,B = C$) which gives us a morphism $!_{I,C} : I \to C$ in $\underline{\mathcal{C}}$ from the initiality of I; (2) take g to be $R\,!_{I,C}$, the restriction of $!_{I,C}$ to $\underline{\mathcal{B}}$.

Def 8. A morphism $g : T \to B$ is *definable by the epi-recursor r* if $g = R\,!_{I,C}$ for some extension C of B.

So an epi-recursor defines a morphism in the base category $\underline{\mathcal{B}}$. However, beyond having the definition go through, we often want to also "remember what happened" in the larger category $\underline{\mathcal{C}}$ because, e.g., properties such as commutation with the non-constructor operators can be useful in themselves.

3.4 Nominal Recursors as Epi-Recursors, Formally

Fig. 2 collects the properties of the operations and relations on terms that are relevant for the recursors—incidentally including some that are generally useful for reasoning about terms. SwVr, SwAp, SwLm relate swapping with the constructors. SwLm points to one of the main appeals of

SwVr	$(\mathsf{Vr}\ x)[z_1 \wedge z_2] = \mathsf{Vr}\ (x[z_1 \wedge z_2])$
SwAp	$(\mathsf{Ap}\ s\ t)[z_1 \wedge z_2] = \mathsf{Ap}\ (s[z_1 \wedge z_2])\ (t[z_1 \wedge z_2])$
SwLm	$(\mathsf{Lm}\ x\ t)[z_1 \wedge z_2]\ =$ $\mathsf{Lm}\ (x[z_1 \wedge z_2])\ (t[z_1 \wedge z_2])$
SwId	$t[z \wedge z] = t$
SwCp	$t[x \wedge y][z_1 \wedge z_2]\ =$ $(t[z_1 \wedge z_2])[(x[z_1 \wedge z_2]) \wedge (y[z_1 \wedge z_2])]$
SwIv	$t[x \wedge y][x \wedge y] = t$
SwFr	if $x \# t$ and $y \# t$ then $t[x \wedge y] = t$
FrSw	$z \# t[x \wedge y]$ if and only if $z[x \wedge y] \# t$
SwFv	if $x, y \notin \mathsf{FV}\ t$ then $t[x \wedge y] = t$
FvSw	$z \in \mathsf{FV}(t[x \wedge y])$ if and only if $z[x \wedge y] \in \mathsf{FV}\ t$
SwCg	if $z \notin \{x_1, x_2\}$ and $z \# t_1, t_2$ and $t_1[z \wedge x_1] = t_2[z \wedge x_2]$ then $\mathsf{Lm}\ x_1\ t_1 = \mathsf{Lm}\ x_2\ t_2$
SwBvr	if $x' \neq x$ and $x' \# t$ then $\mathsf{Lm}\ x\ t = \mathsf{Lm}\ x'\ (t[x' \wedge x])$

RnVr	$(\mathsf{Vr}\ x)[y/z] = \mathsf{Vr}\ (x[y/z])$
RnAp	$(\mathsf{Ap}\ t_1\ t_2)[y/z] = \mathsf{Ap}\ (t_1[y/z])\ (t_2[y/z])$
RnLm_1	if $x \notin \{y, z\}$ then $(\mathsf{Lm}\ x\ t)[y/z] = \mathsf{Lm}\ x\ (t[y/z])$
RnLm_2	$(\mathsf{Lm}\ x\ t)[z/x] = \mathsf{Lm}\ x\ t$
RnCg	if $z \notin \{x_1, x_2\}$ and $z \# t_1, t_2$ and $t_1[z/x_1] = t_2[z/x_2]$ then $\mathsf{Lm}\ x_1\ t_1 = \mathsf{Lm}\ x_2\ t_2$
RnBvr	if $x' \neq x$ and $x' \# t$ then $\mathsf{Lm}\ x\ t = \mathsf{Lm}\ x'\ (t[x'/x])$
RnBvr_2	if $y \neq x'$ then $\mathsf{Lm}\ x\ (t[y/x']) = \mathsf{Lm}\ x'\ (t[y/x'][x'/x])$
RnId	$t[z/z] = t$
RnIm	if $x_1 \neq y$ then $t[x_1/y][x_2/y] = t[x_1/y]$
RnCh	if $y \neq x_2$ then $t[y/x_2][x_2/x_1][x_3/x_2] = t[y/x_2][x_3/x_1]$
RnCm	if $x_2 \neq y_1 \neq x_1 \neq y_2$ then $t[x_2/x_1][y_2/y_1] = t[y_2/y_1][x_2/x_1]$
RnFr	if $y \# t$ then $t[x/y] = t$
FrRn	$z \# t[x/y]$ if and only if $(z = y$ or $z \# t)$ and $(y \# t$ or $x \neq z)$
FrRn_2	$z[x/y] \# t[x/y]$ implies $z \# t$
RnChFr	if $x_2 \# t$ then $t[x_2/x_1][x_3/x_2] = t[x_3/x_1]$

FrVr	if $z \neq x$ then $z \#\ \mathsf{Vr}\ x$
FrAp	if $z \# s$ and $z \# t$ then $z \#\ \mathsf{Ap}\ s\ t$
FrLm	if $z = x$ or $z \# t$ then $z \#\ \mathsf{Lm}\ x\ t$
FvVr	$\mathsf{FV}(\mathsf{Vr}\ x) \subseteq \{x\}$
FvAp	$\mathsf{FV}(\mathsf{Ap}\ t_1\ t_2) \subseteq \mathsf{FV}\ t_1 \cup \mathsf{FV}\ t_2$
FvLm	$\mathsf{FV}(\mathsf{Lm}\ x\ t) \subseteq \mathsf{FV}\ t \setminus \{x\}$

PmVr	$(\mathsf{Vr}\ x)[\sigma] = \mathsf{Vr}\ (\sigma\ x)$
PmAp	$(\mathsf{Ap}\ s\ t)[\sigma] = \mathsf{Ap}\ (s[\sigma])\ (t[\sigma])$
PmLm	$(\mathsf{Lm}\ x\ t)[\sigma] = \mathsf{Lm}\ (\sigma\ x)\ (t[\sigma])$
PmId	$t[\mathsf{id}] = t$
PmCp	$t[\sigma][\tau] = t[\tau \circ \sigma]$
PmFv	if $\mathsf{supp}\ \sigma \cap \mathsf{FV}\ t = \emptyset$ then $t[\sigma] = t$
FvPm	$z \notin \mathsf{FV}(t[\sigma])$ if and only if $z[\sigma^{-1}] \notin \mathsf{FV}\ t$
PmBvr	if $x' \neq x$ and $x' \notin \mathsf{FV}\ t$ then $\mathsf{Lm}\ x\ t = \mathsf{Lm}\ x'\ (t[x' \leftrightarrow x])$

SbVr	$(\mathsf{Vr}\ x)[s/z] =$ $(\mathsf{if}\ x = z\ \mathsf{then}\ s\ \mathsf{else}\ \mathsf{Vr}\ x)$
SbAp	$(\mathsf{Ap}\ t_1\ t_2)[s/z]\ =$ $\mathsf{Ap}\ (t_1[s/z])\ (t_2[s/z])$
SbLm	if $x \neq z$ and $x \# s$ then $(\mathsf{Lm}\ x\ t)[s/z] = \mathsf{Lm}\ x\ (t[s/z])$
SbCg	if $z \notin \{x_1, x_2\}$ and $z \# t_1, t_2$ and $t_1[(\mathsf{Vr}\ z)/x_1] = t_2[(\mathsf{Vr}\ z)/x_2]$ then $\mathsf{Lm}\ x_1\ t_1 = \mathsf{Lm}\ x_2\ t_2$
SbBvr	if $x' \neq x$ and $x' \# t$ then $\mathsf{Lm}\ x\ t = \mathsf{Lm}\ x'\ (t[(\mathsf{Vr}\ x')/x])$
SbId	$t[z/z] = t$
SbIm	if $x_1 \neq y$ then $t[(\mathsf{Vr}\ x_1)/y][s/y] = t[(\mathsf{Vr}\ x_1)/y]$
SbCh	if $y \neq x_2$ then $t[(\mathsf{Vr}\ y)/x_2][(\mathsf{Vr}\ x_2)/x_1][s/x_2] = t[(\mathsf{Vr}\ y)/x_2][s/x_1]$
SbCm	if $x \neq y$, $y \# s$ and $x \# t$ then $t[s/x][t/y] = t[t/y][s/x]$
SbFr	if $y \# t$ then $t[s/y] = t$
FrSb	$z \# t[s/y]$ if and only if $(z = y$ or $z \# t)$ and $(y \# t$ or $z \# s)$
SbChFr	if $x_2 \# t$ then $t[(\mathsf{Vr}\ x_2)/x_1][s/x_2] = t[s/x_1]$

FSupFv	$\mathsf{FV}\ t$ is finite
FvDPm	$\mathsf{FV}\ t = \{x \in \mathsf{Var} \mid \{y \mid t[x \leftrightarrow y] \neq t\}$ is infinite$\}$
FvDSw	$\mathsf{FV}\ t = \{x \in \mathsf{Var} \mid \{y \mid t[x \wedge y] \neq t\}$ is infinite$\}$
FCB	there exists x such that $x \notin \mathsf{FV}(\mathsf{Lm}\ x\ t)$ for all t
FSupFr	$\{x.\ \neg\ x \# t\}$ is finite
FrDSw	$x \# t$ if and only if $\{y \mid t[x \wedge y] \neq t\}$ is finite
FrDRn	$x \# t$ if and only if $\{y \mid t[y/x] \neq t\}$ is finite

Fig. 2. Recursion-relevant properties of operations and relations on terms

the swapping operator for developing the theory of λ-calculus: It shows that swapping commutes with λ-abstraction on terms exactly in the same way as it does for preterms, i.e., is oblivious to the non-injectiveness of λ-abstraction. SwId, SwCp, SwIv are algebraic properties of swapping: identity, compositionality and involutiveness. SwFr and FrSw are properties connecting swapping to freshness (and SwFv and FvSw are their alternative free-variable-based formulations). SwFr says that swapping two fresh variables has no effect on the term. FrSw says that freshness of a variable for a swapped term is equivalent to freshness of the swapped variable for the original term—stating for the freshness predicate a variant of what in nominal logic is called *equivariance*. SwCg is a swapping-based congruence property describing a criterion for the equality of two λ-abstractions. SwBvr is a property allowing the renaming of a λ-bound variable with any fresh variable, again via swapping. SwCg and SwBvr are reminiscent of preterm α-equivalence. Most properties of swapping generalize to corresponding properties of permutation, those listed with "Pm" in their name.

FrVr, FrAp and FrLm relate freshness with the constructors, corresponding to an inductive definition of freshness; and FvVr, FvAp and FvLm are their free-variable counterparts. Note that the "if and only if" versions of FrVr, FrAp and FrLm and the equality versions of FvVr, FvAp and FvLm also hold for terms; though for recursion it is not the stronger, but the weaker versions of properties that lead to stronger definitional principles—since they mean weaker constraints on models.

Like swapping, substitution commutes with the constructors, which is expressed in SbVr, SbAp, SbLm. As shown by SbLm, unlike in the case of swapping, substitution's commutation with λ-abstraction requires a freshness condition. Substitution also enjoys congruence and bound-variable renaming properties similar to those of swapping, as expressed by SbCg and SbBvr, and some algebraic properties, as expressed by SbId, SbIm, SbCh and SbCm. The renaming operator of course enjoys all the properties of substitution; e.g., RnVr, RnAp, RnLm_1 and RnCg are the counterparts of SbVr, SbAp, SbLm and SbCg. One may ask why we bother considering renaming, which is a restriction of substitution; the reason is that, again, for expressive recursors we want *less* structure and *weaker* properties.

The last group in the figure are nominal-logic specific properties. FSupFv states that terms have finite support, i.e., finite set of free variables; it can also be expressed directly in terms of swapping (as in §2.2.1). FvDPm and FvDSw state the definability of free-variables from permutations and (alternatively) from swapping. FCB is the *freshness condition for binders* from the statement of the perm/free recursion theorem (Thm. 1), but with the Barendregt set X removed. FCB is weaker than FvLm since it quantifies existentially rather than universally over the bound variable, though in nominal logic they are equivalent (the "some/any" property [Pitts 2006]). Finally, this last group also includes alternative, freshness-based and renaming-based formulations of some of the above properties. Note that, unlike FrDSw, FrDRn would stay true for terms if we replaced "finite" with "empty".

Each of the properties listed in Fig. 2 is satisfied by the terms with their basic operations and relations, i.e., by the term model $\mathcal{T}r(\Sigma)$ for any signature Σ that contains all the symbols referred to in the property. But we can speak of the corresponding properties in relation to any other Σ-model $\mathcal{M}$, and they may or may not be satisfied by $\mathcal{M}$. For example, when we say that the model $\mathcal{M}$ (with carrier M) satisfies SwCg, we mean the following: For all $m_1, m_2 \in M$ and $x_1, x_2, z \in \mathsf{Var}$, if $z \notin \{x_1, x_2\}$ and $z \#^{\mathcal{M}} m_1, m_2$ and $m_1[z \wedge x_1]^{\mathcal{M}} = m_2[z \wedge x_2]^{\mathcal{M}}$ then $\mathsf{Lm}^{\mathcal{M}}\ x_1\ m_1 = \mathsf{Lm}^{\mathcal{M}}\ x_2\ m_2$.

Given a subset Props of the properties in Fig. 2 and a signature Σ comprising the symbols referred to in Props, any Σ-model satisfying Props will be called a (Σ, Props)-*model*. Now we can (re)formulate nominal recursors as epi-recurors:

Thm 9. Consider the nine choices, for $i \in \{1, \ldots, 9\}$, of tuples $r_i = (\underline{\mathcal{B}}, T, \underline{C}_i, I_i, R_i)$ given by the sets of properties Props_i shown in Fig. 3. (E.g., Props_5 is {SwVr, SwAp, SwLm, SwBvr, FrVr, FrAp, FrLm}.) Namely, we assume that the signature Σ_i consists of all the operation and relation symbols occurring in Props_i, and:

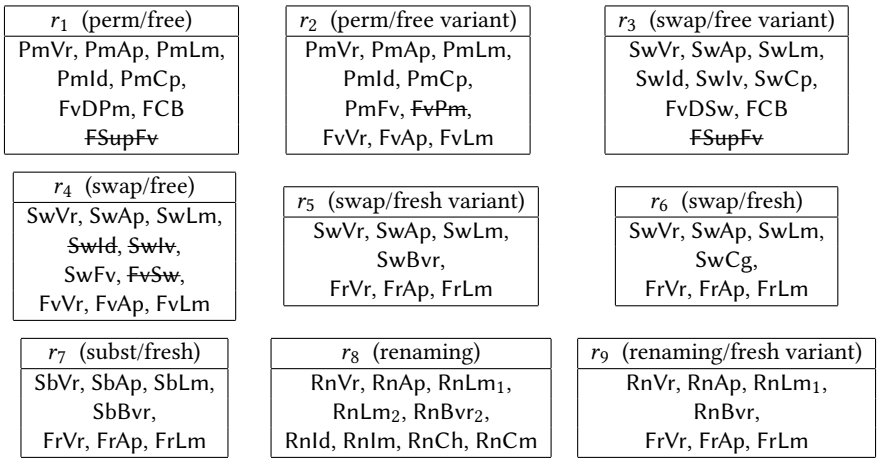

Fig. 3. Sets of properties underlying different nominal recursors. The crossed-out properties FSupFv in r_1 and SwId, SwIv, FvSw in r_4 were in the original recursors but turn out not to be needed.

- $\underline{\mathcal{B}}$ is the category of Σ_{ctor}-models and $T = \mathcal{T}r(\Sigma_{\text{ctor}})$
- $\underline{\mathcal{C}}_i$ is the category of $(\Sigma_i, \text{Props}_i)$-models and I_i is $\mathcal{T}r(\Sigma_i)$
- $R_i : \underline{\mathcal{C}}_i \to \underline{\mathcal{B}}_i$ is the forgetful functor sending $(\Sigma_i, \text{Props}_i)$-models to their underlying Σ_{ctor}-models

Then r_i is an epi-recursor. In particular, $\mathcal{T}r(\Sigma_i)$ is the initial $(\Sigma_i, \text{Props}_i)$-model.

Next we discuss this theorem's nine statements of epi-recursion principles. We distinguish between five "original recursors" from the literature and four "variant recursors" obtained from those.

3.4.1 The Original Recursors. As suggested by the names in Fig. 3, five of these principles, r_1, r_4, r_6, r_7 and r_8, are reformulations of the (stripped down versions of) nominal recursors from §2.2.

This is easy to see in the case of r_6 and r_7. Indeed, after removing the term arguments of the operations and relations, Thms. 3 (swap/fresh) and 4 (subst/fresh) simply state, for a suitable extension of the constructor signature Σ_{ctor}, the initiality of the corresponding term model among all models satisfying Props_6 or Props_7. Moreover, Thm. 5 (about renaming recursion) is easily seen to be exactly r_8.

Seeing that r_1 is the stripped down version of the perm/free recursor (from Thm 1) requires a bit of work. After removing X from (i.e., taking X to be $\emptyset$ in Thm. 1), we see that a nominal set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ together with $\emptyset$-supported operations $\text{Vr}^{\mathcal{A}}$, $\text{Ap}^{\mathcal{A}}$ and $\text{Lm}^{\mathcal{A}}$ can be equivalently described as a $(\Sigma_1, \text{Props}_1)$-model. Moreover, the properties of the unique function $g : \text{Tr} \to A$ guaranteed by Thm. 1 are equivalent to those of Σ_i-morphisms. (Popescu [2023b, App. A] gives details.) Thm. 1 does not actually need the finite-support condition FSupFv for the target domain—which is why in Fig. 3 we show it for r_1 (and for the variant r_3 discussed below) as crossed out.

Seeing that r_4 is the stripped down version of the swap/free recursor (from Thm. 2) is also not immediate. After removing the Barendregt parameterization on X from Thm. 2, we obtain operations and relations that fit the pattern of full-fledged recursion, i.e., iteration plus additional term arguments—e.g., $\text{Vr}^{\mathcal{A}} : \text{Var} \to A$, $\text{Ap}^{\mathcal{A}} : (\text{Tr} \times A) \to (\text{Tr} \times A) \to A$ and $\text{Lm}^{\mathcal{A}} : \text{Var} \to (\text{Tr} \times A) \to A$. So the situation becomes similar to that of r_6 and r_7 versus Thms. 3 and 4. However, two of Thm. 2's assumptions, (2) and (3), do not directly fit the normal full-fledged recursion pattern. But after using that $\text{FV}\ (\text{Ap}\ t_1\ t_2) = \text{FV}\ t_1 \cup \text{FV}\ t_2$ and $\text{FV}\ (\text{Lm}\ x\ t) = \text{FV}\ t \setminus \{x\}$, they are seen equivalent to:

(2) If $\text{FV}^{\mathcal{A}}\ a_1 \subseteq \text{FV}\ t_1$ and $\text{FV}^{\mathcal{A}}\ a_2 \subseteq \text{FV}\ t_2$ then $\text{FV}^{\mathcal{A}}(\text{Ap}^{\mathcal{A}}\ (t_1, a_1)\ (t_2, a_2)) \subseteq \text{FV}\ t_1 \cup \text{FV}\ t_2$

(3) If $\text{FV}^{\mathcal{A}}\ a \subseteq \text{FV}\ t$ then $\text{FV}^{\mathcal{A}}(\text{Lm}^{\mathcal{A}}\ x\ (t, a)) \subseteq \text{FV}\ t \setminus \{x\}$

In this form, they are seen to express a kind of full-fledged recursion that is optimized for the free-variable operator. Indeed, they are weaker versions of ones that *do* fit the pattern:

(2) $\mathsf{FV}^{\mathcal{A}}(\mathsf{Ap}^{\mathcal{A}}\ (t_1, a_1)\ (t_2, a_2)) \subseteq (\mathsf{FV}^{\mathcal{A}}\ a_1 \cup \mathsf{FV}\ t_1) \cup (\mathsf{FV}^{\mathcal{A}}\ a_2 \cup \mathsf{FV}\ t_2)$

(3) $\mathsf{FV}^{\mathcal{A}}(\mathsf{Lm}^{\mathcal{A}}\ x\ (t, a)) \subseteq (\mathsf{FV}^{\mathcal{A}}\ a \cup \mathsf{FV}\ t) \setminus \{x\}$

(In Popescu [2023b, App. D] we show how this free-variable-specific optimization can be seen as a general enhancement available to all our discussed recursors that involve freeness or freshness.) Removing the term arguments from the latter turns them into Fig. 2's FvAp and FvLm; and removing the term arguments from the other assumptions in Thm. 2 turns them into the other properties of Props_4. Finally, the conclusion of Thm. 2 corresponds precisely to the Σ_2-morphism conditions.

Three of the properties originally postulated by Norrish [2004] for the swap/free recursor, SwId, SwIv and FvSw, are not needed, meaning that the recursion theorem holds without them (hence they are crossed out under r_4 in Fig. 3.) This is a surprising result, given the careful analysis done by Norrish when distilling the required properties for his recursor to work. We detected this redundancy while subsuming the $(\Sigma_4, \mathsf{Props}_4)$-models to the more general $(\Sigma_5, \mathsf{Props}_5)$-models during recursor comparison (discussed in §4), so this strengthening owes to the different path taken when proving r_5.

3.4.2 The Variant Recursors. The remaining principles, r_2, r_3, r_5 and r_9, are obtained by combining axioms of the original recursors. They act as bridges between the latter helping their comparison, but are also of independent interest, e.g., r_9 will be seen to be maximal with respect to expressiveness.

We call r_2 a "perm/free variant" because it is another recursor based on permutation and freeness, just like the original perm/free recursor r_1. However r_2 does not follow the nominal-set route of r_1 (which defines the free-variable, i.e., support operator from permutation, via FvDPm) but instead follows the idea of the swap/free recursor r_4 (using permutation instead of swapping) and postulates properties connecting the free-variable operator with permutation (via PmFv and FvPm) and with the constructors (via FvVr, FvAp and FvLm). In short, r_2 is a hybrid between r_1 and r_4. For the symmetry of presentation, under r_2 the figure also shows FvPm—the permutation counterpart of FvSw, but crosses it out because, like FvSw, is also not needed. Another hybrid between the two is the swap-free variant r_3, which uses the swapping operator like r_4 and nominal-set-like axioms like r_1.

r_5 is an r_6–r_7 hybrid, born from the observation that r_6 and r_7 have similar structures, in that they both axiomatize the interaction between constructors and freshness, and between constructors and their specific operator (either swapping or substitution); of course, substitution behaves differently from swapping w.r.t. constructors, but the respective constructor-commuting properties (SbVr, SbAp and SbLm vs. SwVr, SwAp and SwLm) have a similar flavor. The difference between r_6 and r_7 lies in the additional property that they use to further underpin recursion over the constructors: in one case via a congruence rule SwCg and in the other via a bound-variable-renaming rule SbBvr. However, both these latter types of rules make sense for the other operator too, *mutatis mutandis*. As it turns out, we can replace SwCg with SbBvr in the swap/fresh recursor r_6, obtaining the swap/fresh variant r_5. But we cannot perform the dual modification to the subst/fresh recursor r_7, where replacing SbBvr with SbCg would not give a valid recursor; the reason is that, unlike swapping, substitution-like operators need a more delicate handling of the bound variables, which SbBvr but not SbCg can achieve.

Finally, r_9 is a r_7–r_8 hybrid, in that it has axioms similar to r_7, but uses renaming like r_8 rather than substitution. And similarly to the case of substitution, replacing RnBvr with RnCg would not work.

Proof idea for Thm. 9. For any $i \in \{1, \ldots, 9\}$, the only non-trivial part of the statement that r_i is an epi-recursor is the initiality theorem, i.e., the fact that $\mathcal{T}r(\Sigma_i)$ is the initial $(\Sigma_i, \mathsf{Props}_i)$-model.

The initiality theorems for the original recursors already have been proved in the literature (as we discussed in §2), whereas the variant recursors r_2, r_3, r_5 and r_9 are new. We (re)proved all these recursors via the following route: First we gave direct proofs for r_6 and r_9, and then we used the transformations underlying the expressiveness relations in Thm. 12 in order to infer

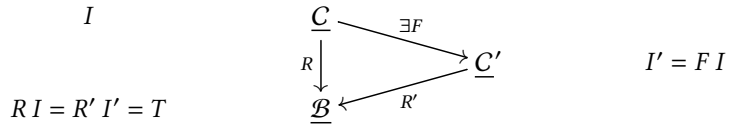

Fig. 4. Criterion for comparing expressiveness

("borrow") the initiality theorems for the others from the above two (which are at the top of Thm. 12's expressiveness hierarchy). Popescu [2023b, App. C] gives details.

Next, we show the proof idea for r_9, which is a generalization/adaptation of that for r_7 from Popescu [2023c]. Let $\mathcal{M}$ be a $(\Sigma_9, \mathsf{Props}_9)$-model. We first define a relation $R : \mathsf{Tr} \to M \to \mathsf{Bool}$, with inductive clauses reflecting the desired properties of commutation with the constructors:

$$R\ (\mathsf{Vr}\ x)\ (\mathsf{Vr}^{\mathcal{M}}\ x) \qquad \frac{R\ t_1\ m_1 \qquad R\ t_2\ m_2}{R\ (\mathsf{Ap}\ t_1\ t_2)\ (\mathsf{Ap}^{\mathcal{M}}\ m_1\ m_2)} \qquad \frac{R\ t\ m}{R\ (\mathsf{Lm}\ x\ t)\ (\mathsf{Lm}^{\mathcal{M}}\ x\ m)}$$

To obtain a Σ_9-morphism $f : \mathsf{Tr} \to M$, it suffices to prove that R (1) is total, (2) is functional, (3) preserves renaming and (4) preserves freshness, since then we can take f to be the function induced by R. Property (1) (totality) follows easily by standard induction on terms. The remaining properties, (2)–(4), follow by a simultaneous inductive proof using a form of "renaming-based induction" on terms: Given a predicate $\varphi : \mathsf{Tr} \to \mathsf{Bool}$, to show $\forall t \in \mathsf{Tr}.\ \varphi\ t$ it suffices to show the following: (i) $\forall x \in \mathsf{Var}.\ \varphi\ (\mathsf{Vr}\ x)$, (ii) $\forall t_1, t_2 \in \mathsf{Tr}.\ \varphi\ t_1\ \&\ \varphi\ t_2 \to \varphi\ (\mathsf{Ap}\ t_1\ t_2)$, and (iii) $\forall x \in \mathsf{Var},\ t \in \mathsf{Tr}.\ (\forall s \in \mathsf{Tr}.\ \mathsf{RConnect}\ t\ s \to \varphi\ s) \to \varphi\ (\mathsf{Lm}\ x\ t)$, where $\mathsf{RConnect}\ t\ s$ means that s is obtained from t by a chain of renamings. (So we take φ to be the conjunction of (2)–(4).) The uniqueness of f follows by induction on terms. The proof for r_6 is similar to that for r_9, but uses a corresponding swapping-based induction. □

4 COMPARING RECURSORS

An advantage of viewing nominal recursors as epi-recursors is clear sight on their relative expressiveness. In this section, we start with a direct means of comparing epi-recursor expressiveness and instantiate it to our nominal recursors (§4.1). Then we analyze a problematic example, semantic interpretation (§4.2), which suggests a gentler comparison—yielding a much flatter expressiveness hierarchy (§4.3). While the kind of relationships we establish show how a recursor can replace another, they do not imply that the converse is not true, and indeed in some cases the converse is true, making the recursors equivalent (w.r.t. a tighter or gentler comparison); but in two cases we also know that the converse is not true, meaning the relation there is strict (§4.4).

4.1 A Head-to-Head Comparison

Def 10. Given epi-recursors $r = (\underline{\mathcal{B}}, T, \underline{C}, I, R)$ and $r' = (\underline{\mathcal{B}}, T, \underline{C}', I', R')$ with the same base category $\underline{\mathcal{B}}$ and base object T, we call r' *stronger than* r, written $r' \geq r$, if r' can define everything that r can, i.e.: for all objects B in $\underline{\mathcal{B}}$ and morphisms $g : T \to B$, g definable by r implies g definable by r'.

It is easy to see that $\geq$ is a preorder on epi-recursors. We write $r \equiv r'$ to state that r and r' have equal strengths, i.e., both $r' \geq r$ and $r \geq r'$ hold. We can establish $r' \geq r$ by showing how to move from r' to r in an initial-object preserving way, as depicted in Fig. 4:

Prop 11. Let $r = (\underline{\mathcal{B}}, T, \underline{C}, I, R)$ and $r' = (\underline{\mathcal{B}}, T, \underline{C}', I', R')$, and assume $F : \underline{C} \to \underline{C}'$ is a prefunctor (i.e., a functor but without the requirement of preserving identity and composition of morphisms) such that $R' \circ F = R$ and $F\ I = I'$. Then $r' \geq r$.

Proof. Assume $g : T \to B$ is definable by r, meaning that $g = R\ !_{I,C}$ for some C in $\underline{C}$. Let $C' = F\ C$. By the initiality of I' and the fact that $F\ I = I'$, we have that $!_{I',C'} = F\ !_{I,C}$. Hence $g = R\ !_{I,C} = R'\ (F\ !_{I,C}) = R'\ !_{I',C'}$, meaning that g is definable by r'. □

(In all our examples, the above initial-object preserving pre-functor condition will be satisfied by actual functors that are left adjoints.) One way to read Prop. 11's criterion (and Fig. 4's picture) is the following: Thinking of R as a kind of "distance" from the extended category $\underline{C}$ (and its initial object I) to the base category $\underline{\mathcal{B}}$ (and the base object B), we have that the smaller this distance, the more expressive the recursor. We have applied this criterion to prove the following expressiveness hierarchy:

Thm 12. The epi-recursors described in Thm. 9 (and in Fig. 3) compare as follows with respect to their expressiveness: $r_6 \geq r_5 \geq r_4 \geq r_2 \geq r_1 \equiv r_3$ and $r_9 \geq r_8, r_7$.

Proof idea. When proving each $r_i \geq r_j$, we instantiate Prop. 11 taking $r' = r_i$ and $r = r_j$. So here $\underline{\mathcal{B}}$ is the category of Σ_{ctor}-models, $\underline{C}'$ that of $(\Sigma_i, \text{Props}_i)$-models, and $\underline{C}$ that of $(\Sigma_j, \text{Props}_j)$-models; R' is the forgetful functor from $(\Sigma_i, \text{Props}_i)$-models to Σ_{ctor}-models, and R the forgetful functor from $(\Sigma_j, \text{Props}_j)$-models to Σ_{ctor}-models; $B = \mathcal{T}r(\Sigma_{\text{ctor}})$, $I' = \mathcal{T}r(\Sigma_i)$ and $I = \mathcal{T}r(\Sigma_j)$. In each case, we must define a pre-functor $F : \underline{C} \to \underline{C}'$ such that $R' \circ F = R$ and $F\ I = I'$. This essentially means showing how to transform $(\Sigma_j, \text{Props}_j)$-models into $(\Sigma_i, \text{Props}_i)$-models in such a manner that $\mathcal{T}r(\Sigma_j)$ becomes $\mathcal{T}r(\Sigma_i)$—which gives F's behavior on objects, while on morphisms F will be the identity. Each time, F will transform models by preserving the carrier set and the constructor-like operators, and possibly defining (1) permutation-like from swapping-like operators or vice versa, (2) freshness-like from free-variable-like operators, or (3) renaming-like from substitution-like operators; these definitions are done just like for concrete terms (where, e.g., we can standardly define freshness from freeness). In each case, the only interesting fact that needs to be checked is that F is well-defined on objects: when starting with a Σ_j-model satisfying Props_j, the result Σ_i-model indeed satisfies Props_i. Everything else amounts to either well-known or trivial properties. Thus, $F\ I = I'$ means that the standard inter-definability properties (1)–(3) hold for terms, e.g., $x \# t$ iff $x \notin \text{FV}\ t$; and $R' \circ F = R$ (i.e., F commutes with the forgetful functors to Σ_{ctor}-models) follows immediately from the fact that F does not change the carrier set or the constructor-like operators. Next, we informally discuss these transformations and highlight the intuitions behind them.

$r_1 \equiv r_3$ holds because permutation-like and swapping-like operators correspond bijectively to each other, allowing one to (functorially) move back and forth between Props_1-models and Props_3-models [Pitts 2013, Section 6.1]. For $r_2 \geq r_1$, we note that r_2 seems *a priori* more flexible than r_1 in that it does not require the free-variable operator to be *definable from* permutation, but only to be *related to* permutation by some weaker properties; and indeed, any $(\Sigma_1, \text{Props}_1)$-model can be proved to be in particular a $(\Sigma_2, \text{Props}_2)$-model. $r_4 \geq r_2$ holds essentially for the same reason why $r_3 \geq r_1$ holds, since the restriction of a permutation to a swapping operator carries over to their axiomatized relationships with free-variable operators, PmFv versus SwFv. (But the converse is not true because r_4 lacks (does not need) some of the swapping axioms that ensure extension to a permutation operator.) $r_5 \geq r_4$ follows using a model transformation that turns the free-variable operator of r_4 into a freshness operator for r_5, using negation; indeed, save for the straightforwardly corresponding FvVr, FvAp, FvLm versus FrVr, FrAp and FrLm, the only difference between r_4 and r_5 is the replacement of SwFv with SwBvr; and the former axiom implies the latter in the presence of FvLm. $r_6 \geq r_5$ follows from the fact that, in the presence of the other axioms in Props_5, SwBvr implies SwCg. $r_9 \geq r_7$ holds because the axioms for substitution imply those for renaming (for the straightforward restriction of a substitution operator to a renaming operator). Finally, the proof of $r_9 \geq r_8$ takes advantage of the fact that, in a constructor-enriched renset (structures axiomatizing renaming that form the basis of recursor r_8), freshness is definable from renaming [Popescu 2023c]. □

Thus, there are two recursors at the top of the expressiveness hierarchy: the swap/fresh recursor r_6 and the renaming/fresh variant recursor r_9. Roughly speaking, these two recursors' expressiveness is strong because their underlying axiomatizations:

- keep freshness only loosely coupled with other operators such as swapping, permutation or renaming—unlike r_1, r_3 and r_8 which ask that freshness be definable from them;
- use congruence or renaming axioms that target exactly the ingredients needed for having recursion go through—unlike those of r_1, r_2, r_3, r_4 and r_8, which employ algebraic axiomatizations such as nominal sets, swapping structures or rensets;
- keep the structure of their operators minimalistic and non-redundant—unlike r_7, whose operator emulates substitution, which is more than needed (since renaming would suffice).

Choosing between swapping and permutation as recursion primitives turned out to be interesting. The two are known to be equivalent for nominal sets [Pitts 2013, §6.1], as reflected by $r_1 \equiv r_3$.

But they are no longer equivalent when loosening the axiomatization to include freshness as a primitive—as reflected by the fact that $r_4 \geq r_2$ but (as we will show in §4.4) not vice versa. This is because the proof of the r_4 recursor (by Norrish [2004]) gets away without assuming swapping compositionality SwCp, which is a crucial ingredient for extending swapping to permutation. Moreover, in an indirect way, we also showed the other crucial ingredients needed for this extension, namely SwId and SwIv, are not required for recursion either. Thus, in this case swapping-based recursion requires significantly weaker assumptions than permutation-based recursion.

4.2 Semantic-Interpretation Example

The notion of interpreting syntax in semantic domains is a well-known challenging example for binding-aware recursion. Let D be a set and $\mathsf{AP} : D \to D \to D$ and $\mathsf{LM} : (D \to D) \to D$ be operators modeling semantic notions of application and abstraction. (Subject to some axioms that are not of interest here, the structure $(D, \mathsf{AP}, \mathsf{LM})$ is known as a Henkin model for λ-calculus [Barendregt 1985].) An environment will be a function $\xi : \mathsf{Var} \to D$. Given $x, y \in \mathsf{Var}$ and $d, e \in D$, we write $\xi\langle x := d\rangle$ for ξ updated with value d for x, and write $\xi\langle x := d, y := e\rangle$ instead of $\xi\langle x := d\rangle\langle y := e\rangle$.

The semantic interpretation $\mathsf{sem} : \mathsf{Tr} \to (\mathsf{Var} \to D) \to D$ should go recursively by the clauses:

(1) $\mathsf{sem}\ (\mathsf{Vr}\ x)\ \xi = \xi\ x$ (2) $\mathsf{sem}\ (\mathsf{Ap}\ t_1\ t_2)\ \xi = \mathsf{AP}\ (\mathsf{sem}\ t_1\ \xi)\ (\mathsf{sem}\ t_2\ \xi)$

(3) $\mathsf{sem}\ (\mathsf{Lm}\ x\ t)\ \xi = \mathsf{LM}\ (d \mapsto \mathsf{sem}\ t\ (\xi\langle x := d\rangle))$

Of course, these clauses do not work out of the box (i.e., do not form a correct recursive definition yet), and here is where the nominal recursors can help. First, let us attempt to deploy the perm/free recursor r_1. To this end, we try to organize the target domain $I = (\mathsf{Var} \to D) \to D$ as a $(\Sigma_1, \mathsf{Props}_1)$-model $\mathcal{I}$. The three desired clauses above already determine constructor operations $\mathsf{Vr}^{\mathcal{I}}$, $\mathsf{Ap}^{\mathcal{I}}$ and $\mathsf{Lm}^{\mathcal{I}}$ on the set of interpretations, $I = (\mathsf{Var} \to D) \to D$, namely:

(1) $\mathsf{Vr}^{\mathcal{I}} : \mathsf{Var} \to I$ by $\mathsf{Vr}^{\mathcal{I}}\ x\ i\ \xi = \xi\ x$ (2) $\mathsf{Ap}^{\mathcal{I}} : I \to I \to I$ by $\mathsf{Ap}^{\mathcal{I}}\ i_1\ i_2\ \xi = \mathsf{AP}\ (i_1\ \xi)\ (i_2\ \xi)$

(3) $\mathsf{Lm}^{\mathcal{I}} : \mathsf{Var} \to I \to I$ by $\mathsf{Lm}^{\mathcal{I}}\ x\ i\ \xi = \mathsf{LM}\ (d \mapsto i\ (\xi\langle x := d\rangle))$

Thus, we already have the Σ_{ctor} component of our intended model. Now we must define a permutation operator on I. The definition is obtained by analyzing the desired behavior of the to-be-defined function sem w.r.t. permutation; i.e., determining the value of $\mathsf{sem}(t[\sigma])$ from $\mathsf{sem}\ t$ and σ. The answer is (4) $\mathsf{sem}\ (t[\sigma])\ \xi = \mathsf{sem}\ t\ (\xi \circ \sigma)$, and leads to defining $\_[\_]^{\mathcal{I}}$ by $i\ [\sigma]^{\mathcal{I}}\ \xi = i\ (\xi \circ \sigma)$.

Note that, towards the goal of building a $(\Sigma_1, \mathsf{Props}_1)$-model $\mathcal{I}$, we had no other choice on defining the operators $\mathsf{Vr}^{\mathcal{I}}$, $\mathsf{Ap}^{\mathcal{I}}$, $\mathsf{Lm}^{\mathcal{I}}$ and $[\_]^{\mathcal{I}}$ on the target domain I. And the free-variable (support) operator $\mathsf{FV}^{\mathcal{I}}$ is also uniquely determined by the axiom FvDPm (definability of freeness from permutation).

Finally, to deploy r_1 and obtain a function sem satisfying clauses (1)–(4), it remains to check that $\mathcal{I}$ satisfies Props_1. But, as it turns out, $\mathcal{I}$ does not satisfy one of the axioms in Props_1, namely FCB (freshness condition for binders). Indeed, FCB requires that there exists a variable x such that for all

$i \in I$, $x \notin \mathsf{FV}^{\mathcal{I}}\,(\mathsf{Lm}^{\mathcal{I}}\ x\ i)$. Applying FvDPm and the definitions of $\_[\_]^{\mathcal{I}}$ and $\mathsf{Lm}^{\mathcal{I}}$, we see that $x \notin \mathsf{FV}^{\mathcal{I}}\,(\mathsf{Lm}^{\mathcal{I}}\ x\ i)$ means $\mathsf{LM}\ (d \mapsto i\ (\xi\langle x := d, y := \xi\ x\rangle)) = \mathsf{LM}\ (d \mapsto i\ (\xi\langle x := d\rangle))$ holds for all but a finite number of variables y. The only chance for the above to be true is if i, when applied to an environment, say ξ', ignores the value of y in ξ' for all but a finite number of variables y; in other words, i only analyzes the values of a finite number of variables in ξ'—but this is not guaranteed to hold for arbitrary elements $i \in I$. Thus, r_1 cannot be deployed directly to define semantic interpretations.

Other recursors in our list can. E.g., the perm-free variant r_2 can be deployed as follows. We use the same definitions for $\mathsf{Vr}^{\mathcal{I}}$, $\mathsf{Ap}^{\mathcal{I}}$, $\mathsf{Lm}^{\mathcal{I}}$ and $[\_]^{\mathcal{I}}$, but now we can choose the free-variable operator $\mathsf{FV}^{\mathcal{I}}$ more flexibly, making sure that the $(\Sigma_2, \mathsf{Props}_2)$-morphism condition holds for $\mathsf{FV}^{\mathcal{I}}$ versus FV, i.e., that (5) $\mathsf{FV}^{\mathcal{I}}(\mathsf{sem}\ t) \subseteq \mathsf{FV}\ t$ holds. Namely, we define $\mathsf{FV}^{\mathcal{I}}\,i$ as $\{x \in \mathsf{Var} \mid \exists \xi : \mathsf{Var} \to D,\ d \in D.\ i\ \xi \neq i\ (\xi\langle x := d\rangle)\}$. The definition identifies a natural notion of what it means for a variable to "occur freely" in a semantic item $i \in I$: when i actually depends on x, i.e., when changing the value of x in an input environment ξ makes a difference in the result of applying i. And indeed, with $\mathsf{FV}^{\mathcal{I}}$ defined like this, $\mathcal{I}$ forms a $(\Sigma_2, \mathsf{Props}_2)$-model, which gives us a unique function sem satisfying (1)–(5).

Thus, semantic interpretation is an example where our "head-to-head" comparison has a visible outcome. But there is still an unexplored nuance here, which we discuss next.

Above, we argued that the semantic-interpretation example cannot be defined *directly* using the perm/free recursor r_1. However, as discussed by Pitts [2006, §6.3], it turns out that it can be defined in a more roundabout manner, after some technical hassle. The trick is to restrict the target domain I to a subset I' on which the above defined operators do form an $(\Sigma_1, \mathsf{Props}_1)$-model, and use r_1 to define $\mathsf{sem} : \mathsf{Tr} \to I'$. Namely, I' is defined as $\{i \in I \mid \exists V \subseteq \mathsf{Var}.\ V$ finite and $\forall x \in V.\ \forall \xi, d.\ i\ \xi = i\ (\xi\langle x := d\rangle)\}$. Then one proves that I' is closed under the constructors $\mathsf{Vr}^{\mathcal{I}}$, $\mathsf{Ap}^{\mathcal{I}}$, $\mathsf{Lm}^{\mathcal{I}}$. Moreover, for I' the above problem with FCB disappears, roughly because all the elements of I' are finitary. So I', with the same operators as those we tried for I, now forms a $(\Sigma_1, \mathsf{Props}_1)$-model, and r_1 recursion can proceed and define $\mathsf{sem} : \mathsf{Tr} \to I'$, hence also $\mathsf{sem} : \mathsf{Tr} \to I$.

Having different nominal recursors in front of us laid out as epi-recursors, we can view Pitts's trick in a new light. Remember that, when deploying r_2 to define sem, we used the operator $\mathsf{FV}^{\mathcal{I}}$, which is a laxer notion of free-variable than that allowed by r_1. An equivalent definition of I' is as the set of all elements of I that have $\mathsf{FV}^{\mathcal{I}}$ finite. Thus, Pitts's trick can be seen as borrowing the free-variable operator from the different recursor r_2, in order to single out a suitable target domain for deploying r_1! One can also prove that, on I', the nominal-logic support (defined from permutation via FvDPm) *coincides* with $\mathsf{FV}^{\mathcal{I}}$—which means that, for the target domain I', r_1 works as well as r_2.

Thus, on a subset of the target domain that is closed under constructors, the previously deemed weaker recursor r_1 can simulate r_2. As it turns out, this is a general phenomenon, which we can phrase for epi-recursors as a gentler expressiveness comparison.

4.3 A Gentler Comparison

Our relation $r' \geq r$ compares the strength of epi-recusors directly, as inclusion between what r can define and what r' can define. The discussion ending §4.2 suggests that this relation may be too strict. More flexibly, we could check if what r can define is obtainable from what r' can define *up to composition with a morphism* (which can be an inclusion, as in Pitts's trick).

Formalizing this for two epi-recursors $r = (\underline{\mathcal{B}}, T, \underline{C}, I, R)$ and $r' = (\underline{\mathcal{B}}, T, \underline{C}', I', R')$ must make sure to avoid pathological dependencies. Indeed, a first attempt is: For all objects B in $\underline{\mathcal{B}}$ and morphisms $g : T \to B$, if g is definable by r then there exists an object B_0 and two morphisms $g_0 : T \to B_0$ and $h : B_0 \to B$ such that g_0 is definable by r' and $g = h \circ g_0$. But this would yield a vacuous concept, rendering any epi-recursor r' stronger than any other r: just take $B_0 = T$, $g_0 = 1_T$ (which is obviously definable by r') and $h = g$. So we should be careful not to allow the above

"transition" morphism h to depend on the r-definability morphism g. Otherwise, we would use r-definability itself to reduce r-definability to r'-definability.

For producing morphisms to objects B of $\underline{\mathcal{B}}$ independently of other data, the following concept comes handy. An *initial segment* of a category $\underline{C}$ is a pair $(\underline{C}_0, (m(C) : o(C) \to C)_{C \in \mathrm{Obj}(\underline{C})})$ where $\underline{C}_0$ is a full subcategory of $\underline{C}$ and, for each object C of $\underline{C}$, $o(C)$ is an object of $\underline{C}_0$ and $m(C)$ a morphism in $\underline{C}$. Using an ordering metaphor, an initial segment of a category provides a "smaller" object for any of its objects. Now we can formulate our gentler relation for comparing strength, called quasi-strength:

Def 13. $r' = (\underline{\mathcal{B}}, T, \underline{C}', I', R')$ is *quasi-stronger* than $r = (\underline{\mathcal{B}}, T, \underline{C}, I, R)$, written $r' \gtrsim r$, when there exists an initial segment $(\underline{\mathcal{B}}_0, (m(B) : o(B) \to B)_{B \in \mathrm{Obj}(\underline{\mathcal{B}})})$ of $\underline{\mathcal{B}}$ such that, for all $g : T \to B$ definable by r, there exists a morphism $g_0 : T \to o(B)$ such that g_0 is definable by r' and $g = m(B) \circ g_0$.

Thus, $r' \gtrsim r$ says that what r can define is obtainable from what r' can define up to composition with a morphism that only depends on the target object in the base category. Note that we use initial segments to make sure that the morphisms that "fill the gap" between the two recursors r and r' are given *before hand*, so that they are independent from any specific recursively defined function (in particular, preventing bogus expressiveness orderings like the one exemplified above).

$\gtrsim$ is a preorder weaker than $\geq$. We write $r \cong r'$ to mean that $r' \gtrsim r$ and $r \gtrsim r'$, i.e., r and r' have quasi-equal strengths.

While being a reasonable weakening of $\geq$, the relation $\gtrsim$ is likely to be more costly to deploy than $\geq$. Indeed, as suggested by our discussion in §4.2, applying $r' \gtrsim r$, i.e., using r' in lieu of r, in particular extracting $o(B)$ from B and using $o(B)$ as a "more precise" target domain, can involve non-negligible formal bureaucracy in concrete situations.

Our effective criterion for checking $\geq$ (Prop. 11) can be generalized to deal with $\gtrsim$. Given two categories $\underline{C}$ and $\underline{C}'$, each with initial segments $(\underline{C}_0, (m(C) : o(C) \to C)_{C \in \mathrm{Obj}(\underline{C})})$ and $(\underline{C}'_0, (m'(C) : o'(C) \to C)_{C \in \mathrm{Obj}(\underline{C}')})$, a functor $G : \underline{C} \to \underline{C}'$ is said to *preserve* the indicated initial segments if $G\ o(C) = o'(G\ C)$ and $G\ (m(C)) = m'(G\ C)$ for all $C \in \mathrm{Obj}(\underline{C})$.

Prop 14. Let $r = (\underline{\mathcal{B}}, T, \underline{C}, I, R)$ and $r' = (\underline{\mathcal{B}}, T, \underline{C}', I', R')$. Assume $(\underline{\mathcal{B}}_0, (m(B) : o(B) \to B)_{B \in \mathrm{Obj}(\underline{\mathcal{B}})})$ is an initial segment of $\underline{\mathcal{B}}$ and $(\underline{C}_0, (m_1(C) : o_1(C) \to C)_{C \in \mathrm{Obj}(\underline{C})})$ is an initial segment of $\underline{C}$ such that $\underline{C}_0$ contains I and R preserves the above initial segments, and $F : \underline{C}_0 \to \underline{C}'$ is a pre-functor such that $F\ I = I'$ and $R' \circ F = R_{\restriction \underline{C}_0}$ (where $R_{\restriction \underline{C}_0}$ is the restriction of R to $\underline{C}_0$). Then $r' \gtrsim r$.

The gist of this criterion (and also its proof idea) is shown in Fig. 5: We start with a morphism g definable by r and use the two initial segments to factor it as a morphism g_0 definable by r' and a remainder morphism $m(B)$.

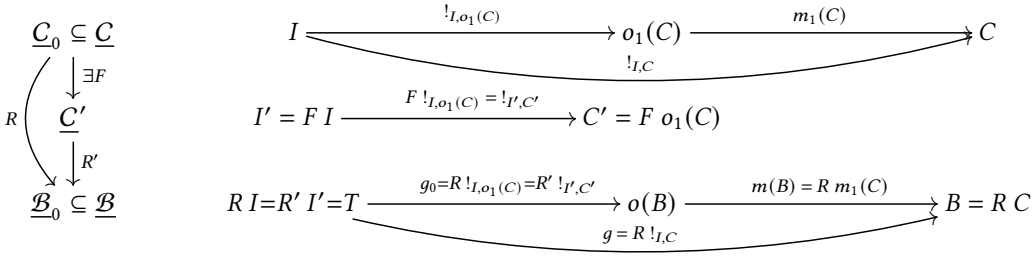

Fig. 5. Gentler criterion for comparing expressiveness

Applying the gentler comparison to our recursors (via Prop. 14) yields a quite surprising result:

Thm 15. The epi-recursors described in Thm. 9 (and in Fig. 3) compare as follows by quasi-strength: $r_1 \cong r_2 \cong r_3 \cong r_4 \cong r_5 \cong r_6 \gtrsim r_8 \cong r_9 \gtrsim r_7$.

Proof idea. When proving each $r_i \gtrsim r_j$, we instantiate Prop. 14 taking $r' = r_i$ and $r = r_j$. So here $\underline{\mathcal{B}}$ is the category of Σ_{ctor}-models, $\underline{\mathcal{C}}'$ that of $(\Sigma_i, \text{Props}_i)$-models, and $\underline{\mathcal{C}}$ that of $(\Sigma_j, \text{Props}_j)$-models; R' is the forgetful functor from $(\Sigma_i, \text{Props}_i)$-models to Σ_{ctor}-models, and R the forgetful functor from $(\Sigma_j, \text{Props}_j)$-models to Σ_{ctor}-models; $B = \mathcal{T}r(\Sigma_{\text{ctor}})$, $I' = \mathcal{T}r(\Sigma_i)$ and $I = \mathcal{T}r(\Sigma_j)$.

We define the initial segment $(\underline{\mathcal{B}}_0, (m(\mathcal{A}) : o(\mathcal{A}) \to \mathcal{A})_{\mathcal{A} \in \text{Obj}(\underline{\mathcal{B}})})$ of $\underline{\mathcal{B}}$ as follows: For any Σ_{ctor}-model $\mathcal{A}$ we take $o(\mathcal{A})$ to be its minimal submodel (subalgebra), i.e., the one generated by $\text{Vr}^{\mathcal{A}}$, $\text{Ap}^{\mathcal{A}}$ and $\text{Lm}^{\mathcal{A}}$; we take $m(\mathcal{A}) : o(\mathcal{A}) \to \mathcal{A}$ to be the inclusion morphism; and we take $\underline{\mathcal{B}}_0$ to be the full subcategory given by constructor-generated models. Each time, we will define the initial segment $(\underline{\mathcal{C}}_0, (m_1(\mathcal{M}) : o_1(\mathcal{M}) \to \mathcal{M})_{\mathcal{M} \in \text{Obj}(\underline{\mathcal{C}})})$ so that, for each $(\Sigma_j, \text{Props}_j)$-model $\mathcal{M}$, $o_1(\mathcal{M})$ is a submodel of $\mathcal{M}$ whose carrier is generated by the constructors ($\text{Vr}^{\mathcal{M}}$, $\text{Ap}^{\mathcal{M}}$ and $\text{Lm}^{\mathcal{M}}$) and will have the other operators from Σ_j defined in specific ways; and $\underline{\mathcal{C}}_0$ will be the full subcategory given by the objects $o_1(\mathcal{M})$. This way, it will be guaranteed that R preserves initial segments.

To prove the $\cong$-chain going from r_1 to r_6, thanks to Thm. 12 and the fact that $\gtrsim$ is weaker than $\geq$, it suffices to prove $r_3 \gtrsim r_6$. We proceed as follows: Given a $(\Sigma_6, \text{Props}_6)$-model $\mathcal{M}$ of carrier M, we take $o_1(\mathcal{M})$ to be a submodel $\mathcal{M}'$ of $\mathcal{M}$, having as carrier set the subset M' of M generated by the constructors $\text{Vr}^{\mathcal{M}}$, $\text{Ap}^{\mathcal{M}}$ and $\text{Lm}^{\mathcal{M}}$, having the constructors and swapping operators inherited from $\mathcal{M}$ and having freshness defined from swapping in nominal style (as in FrDSw); crucially, this definition of freshness turns out to be equivalent to an inductive one using FrVr, FrAp and FrLm, making $\mathcal{M}'$ the minimal $(\Sigma_6, \text{Props}_6)$-submodel of $\mathcal{M}$. Now, the pre-functor F is defined on objects as follows: $F\ \mathcal{M}'$ is the Σ_3-model having the same constructors and swapping operator as $\mathcal{M}'$, and having the free-variable operator defined standardly from the freshness operator of $\mathcal{M}'$, via negation. (And on morphisms, F is the identity.) $F\ \mathcal{M}'$ satisfies Props_3: SwVr, SwAp, SwLm and FvDSw hold by construction, and FCB, SwId, SwIv and SwCp follow by induction on the definition of M'. The other required properties are trivial, e.g., $F\ I = I'$ here means that the standard definition of free-variables from freshness is correct for terms; and $R' \circ F = R_{\restriction \underline{\mathcal{C}}_0}$ means that F commutes with the forgetful functors.

To prove the $(\cong, \gtrsim)$-chain going from r_6 to r_7, again thanks to Thm. 12 it suffices to prove $r_8 \gtrsim r_9$ and $r_6 \gtrsim r_8$. (We will no longer show explicitly the definitions of the initial segment and the pre-functor, but give the ingredients from which they can be constructed similarly to how we did above.) For $r_8 \gtrsim r_9$, we start similarly to the proof of $r_3 \gtrsim r_6$, namely for a $(\Sigma_9, \text{Props}_9)$-model $\mathcal{M}$ we take the minimal submodel $\mathcal{M}'$ where freshness definable from renaming (via FrDRn) turns out to coincide with the inductively defined version via FrVr, FrAp and FrLm. Because the carrier M' of $\mathcal{M}'$ is the image of the unique Σ_9-morphism $f : \mathcal{T}r(\Sigma_9) \to \mathcal{M}$ ensured by the initiality of $\mathcal{T}r(\Sigma_9)$, $\mathcal{M}'$ satisfies all unconditional equations satisfied by $\mathcal{T}r(\Sigma_9)$, in particular, all the Props_8 properties.

Finally, the proof of $r_6 \gtrsim r_8$ exploits the observation that renaming is definable from swapping not only for terms, but also for any $(\Sigma_8, \text{Props}_8)$-model $\mathcal{M}$ that guarantees the existence of fresh variables, i.e., having its elements finitely supported: $m\ [z_1 \wedge^{\mathcal{M}} z_2]$ is defined as $m\ [y/^{\mathcal{M}} z_1]\ [z_1/^{\mathcal{M}} z_2]\ [z_2/^{\mathcal{M}} y]$ where y is fresh (and, using the Props_8 axioms, the choice of y can be proved not to matter). While arbitrary $(\Sigma_8, \text{Props}_8)$-models $\mathcal{M}$ do not guarantee finite support, we can again switch to a minimal submodel $\mathcal{M}'$ that does guarantee it—and in $\mathcal{M}'$ the above definition indeed yields a swapping operator that together with the constructors and freshness satisfies Props_6. □

Thus, $\gtrsim$ brings a dramatic flattening of the $\geq$ hierarchy established by Thm. 12: All the swapping- and permutation-based recursors r_1–r_6 have equal quasi-strengths. The intuition for this, as we discovered during the proofs, is the following. Recall that the differences in strength (using $\geq$) between these recursors were due to: (1) looseness or tightness of their connection between swapping/permutation and freeness/freshness, (2) higher flexibility of swapping compared to permutation, and (3) more focused nature of congruence compared to an algebraic axiomatization. Remarkably, all these differences vanish if we are allowed to navigate along submodels, which $\gtrsim$ enables. This is because

(as explained in the proof of Thm. 15), certain minimal submodels are much more "term-like" than an arbitrary model; they generalize Pitts's submodel definition for semantic interpretation, where nominal-style freshness coincides with other, more loosely axiomatized notions of freshness.

An interesting takeover when switching from $\geq$ to $\gtrsim$ is the swapping/permutation-based recursors r_1–r_6 becoming (quasi-)stronger than the renaming-based recursors r_8 and r_9. Indeed, defining renaming from swapping or vice versa seems impossible in arbitrary models, meaning these two types of recursors are $\geq$-incomparable. But when switching to submodels (allowed by $\gtrsim$) one direction is possible: The swapping of two variables can be defined in a renaming-based model similarly to how it is done for concrete terms, via picking an intermediate fresh variable; and "picking fresh" is possible in minimal submodels because everything there is finitely supported.

Summary. Epi-recursors are comparable for expressiveness by a strict relation $\geq$, saying that everything definable by one is definable by the other, and a laxer relation $\gtrsim$, saying that everything definable by one can be defined by the other with the help of an additional morphism, typically a submodel inclusion. The handling of the semantic-interpretation example with the nominal-logic recursor was our inspiration for $\gtrsim$, and suggests the additional overhead incurred by $\gtrsim$. The $\gtrsim$-hierarchy is significantly flatter than the $\geq$-hierarchy, sending an egalitarian message: *Most nominal recursors turn out to have the same strength*, with the only nuance that those based on symmetric operators (swapping and permutation) are more expressive than those based on asymmetric ones (renaming and substitution).

4.4 Negative Results

Thms. 12 and 15 establish $\geq$ and $\gtrsim$ relationships between recursors, which essentially tell us that a recursor can replace/simulate another recursor (under a tighter or a looser notion of replacement). But how about the question of when a recursor *cannot* replace another? The discussion in §4.2 suggests that $r_1 \geq r_2$ does not hold. The next proposition states the two negative results we know so far:

Prop 16. $r_1 \not\geq r_2$ (i.e., it is not the case that $r_1 \geq r_2$) and $r_2 \not\geq r_4$ (i.e., it is not the case that $r_2 \geq r_4$).

Proof sketch. To prove $r_i \not\geq r_j$, we must provide a $(\Sigma_j, \mathsf{Props}_j)$-model $\mathcal{M}$ for which the Σ_{ctor}-reduct (i.e., the Σ_{ctor}-model obtained by forgetting the operators from $\Sigma_j \setminus \Sigma_{\mathsf{ctor}}$) cannot be the Σ_{ctor}-reduct of any $(\Sigma_i, \mathsf{Props}_i)$-model.

For $r_1 \not\geq r_2$, we take the $(\Sigma_2, \mathsf{Props}_2)$-model $\mathcal{M}$ to have as carrier the set $M = \mathsf{Tr} \cup A$, where A consist of all the streams of variables (in $\mathsf{Var}^{\mathbb{N}}$) whose sets of occurring variables are infinite. We let $(t_i)_{i \in \mathbb{N}}$ be a family of terms such that all are ground ($\mathsf{FV}\ t_i = \emptyset$) and mutually distinct. We define $\mathcal{M}$'s operators on M by extending the standard term operators from Tr as follows, for any $xs \in A$ (where map_σ is the standard stream-map operator and $\mathsf{rem}_y\ xs$ removes all occurrences of y from xs):

- $\mathsf{FV}^{\mathcal{M}} xs = \mathsf{Vars}\ xs$
- $\mathsf{Lm}^{\mathcal{M}}\ y\ xs = \mathsf{rem}_y\ xs$ for any $y \in \mathsf{Var}$
- $\mathsf{Ap}^{\mathcal{M}}\ xs\ t_i = \mathsf{Vr}\ xs_i$ for any $i \in \mathbb{N}$
- $\mathsf{Ap}^{\mathcal{M}}\ xs\ m = t_0$ for any $m \in M \setminus \{t_i \mid i \in \mathbb{N}\}$
- $\mathsf{Ap}^{\mathcal{M}}\ s\ xs = t_0$ for any $s \in \mathsf{Tr}$
- $xs[\sigma]^{\mathcal{M}} = \mathsf{map}_\sigma\ xs$ for any $\sigma \in \mathsf{Perm}$

Note that, on A, the free-variable-like and abstraction-like operators are natural, in particular $\mathsf{Lm}^{\mathcal{M}}$ removes all occurrences of the abstracted variable. On the other hand, the application-like operator is contrived: the only interesting case is $\mathsf{Ap}^{\mathcal{M}}\ xs\ t_i$, where application emulates the i'th projection, retrieving the i'th element of the stream xs; in the other cases application simply returns the ground term t_0. We can check that $\mathcal{M}$ thus defined satisfies the Props_2 properties. One the other hand, the Σ_{ctor}-reduct of $\mathcal{M}$, i.e., $\mathsf{Tr} \cup A$ equipped with the above-defined constructor-like operators, cannot be the reduct of any $(\Sigma_1, \mathsf{Props}_1)$-model, i.e., there is no way to define the operators $\_[\_]'$ and FV' on $\mathsf{Tr} \cup A$ that, together with $\mathsf{Vr}^{\mathcal{M}}$, $\mathsf{Ap}^{\mathcal{M}}$ and $\mathsf{Ap}^{\mathcal{M}}$, make it a $(\Sigma_1, \mathsf{Props}_1)$-model. Indeed, if such operators $\_[\_]'$ and FV' existed, then the Props_1 axioms would imply that $\_[\_]'$ extends the standard permutation operators from Tr and A, and then that $\mathsf{FV}'\ xs = \mathsf{Var}$ for all xs, which contradicts FCB.

For $r_2 \not\geq r_4$, we take the $(\Sigma_4, \mathsf{Props}_4)$-model $\mathcal{M}$ to have as carrier the set $M = \mathsf{Tr} \cup \{a\}$ (where $a \notin \mathsf{Tr}$), i.e., to consist of terms plus an additional element a. Let x be a fixed variable. We define $\mathcal{M}$'s operators on M by extending the standard term operators from Tr as follows:

- $\mathsf{FV}^{\mathcal{M}} a = \mathsf{Var}$ (the set of all variables)
- $\mathsf{Lm}^{\mathcal{M}}\ y\ a = \mathsf{Lm}^{\mathcal{M}}\ y\ (\mathsf{Vr}\ x)$ for any $y \in \mathsf{Var}$
- $\mathsf{Ap}^{\mathcal{M}}\ a\ a = \mathsf{Ap}^{\mathcal{M}}\ (\mathsf{Vr}\ x)\ (\mathsf{Vr}\ x)$
- $\mathsf{Ap}^{\mathcal{M}}\ a\ t = \mathsf{Ap}^{\mathcal{M}}\ (\mathsf{Vr}\ x)\ t$ for any $t \in \mathsf{Tr}$
- $\mathsf{Ap}^{\mathcal{M}}\ t\ a = \mathsf{Ap}^{\mathcal{M}}\ t\ (\mathsf{Vr}\ x)$ for any $t \in \mathsf{Tr}$
- $a[z_1 \wedge z_2]^{\mathcal{M}} = \mathsf{Vr}\ (x[z_1 \wedge z_2])$ for any $z_1, z_2 \in \mathsf{Var}$

Thus, the free variables of a are the entire set of variables, and the constructor and swapping operators on a yield the same results as for $\mathsf{Vr}\ x$, i.e., have $\mathsf{Vr}\ x$ act in lieu of a. We can check that $\mathcal{M}$ satisfies Props_4. On the other hand, the Σ_{ctor}-reduct of $\mathcal{M}$, i.e., $\mathsf{Tr} \cup \{a\}$ equipped with the above-defined constructor-like operators, cannot be the reduct of any $(\Sigma_2, \mathsf{Props}_2)$-model, i.e., there is no way to define the operators $\_[\_]'$ and FV' on $\mathsf{Tr} \cup \{a\}$ that, together with $\mathsf{Vr}^{\mathcal{M}}$, $\mathsf{Ap}^{\mathcal{M}}$ and $\mathsf{Ap}^{\mathcal{M}}$, make it a $(\Sigma_2, \mathsf{Props}_2)$-model. Indeed, if such operators $\_[\_]'$ and FV' existed, then the axioms in Props_2 would imply that $\_[\_]'$ extends the standard permutation operator on Tr, and also that $\_[\sigma]'$ is bijective on $\mathsf{Tr} \cup \{a\}$ for any permutation σ; so the only possibility is that $a[\sigma]' = a$ for any σ; this together with PmAp would imply that $(\mathsf{Ap}^{\mathcal{M}}\ a\ a)[\sigma]' = \mathsf{Ap}^{\mathcal{M}}\ (a[\sigma]')\ (a[\sigma]') = \mathsf{Ap}^{\mathcal{M}}\ a\ a$, i.e., $(\mathsf{Ap}\ (\mathsf{Vr}\ x)\ (\mathsf{Vr}\ x))[\sigma] = \mathsf{Ap}\ (\mathsf{Vr}\ x)\ (\mathsf{Vr}\ x)$, which is false for any σ that modifies x. □

Note that, if we write $>$ for the strict version of $\geq$ (defined as $r > r'$ iff $r \geq r'$ and $r' \not\geq r$), then assuming $r \geq r'$, a negative result $r' \not\geq r$ is a strictness result $r > r'$. So from Thm. 12 , Prop. 16 and Thm. 15 we have $r_2 > r_1$ but $r_2 \cong r_1$, and also $r_4 > r_2$ but $r_4 \cong r_2$. We do not yet have negative/strictness results across the board, in particular, none for $\gtrsim$.

5 THE COINDUCTIVE SPECTRUM

Next we will shift focus from the standard terms with bindings discussed so far, which were defined inductively, to (possibly) infinitary non-well-founded terms with bindings, defined *coinductively*, where the constructors can be applied an infinite number of times. Unlike with the inhabitants of standard coinductive datatypes, we will still identify terms modulo α-equivalence. Rather than recursion, we will now study *corecursion*, that is, mechanisms for defining functions having terms not as source domain, but as target domain (codomain). Building on the experience of having handled the recursors, we will now take a more direct route, and at a faster pace: After recalling infinitary terms (§5.1), we introduce abstract epi-corecursors (§5.2), then delve into the spectrum of nominal corecursor instances, connect with pre-existing nominal corecursors, and establish a hierarchy (§5.3).

5.1 Infinitary Terms with Bindings

Let Var be a set of variables whose cardinality is $\aleph_1$, the first uncountable cardinal. (Any uncountable regular cardinal would do—we only care about the existence of fresh variables for any term.) The set Tr_∞ of *infinitary λ-terms*, *iterms* for short, is defined by the same grammar as before, $t ::= \mathsf{Vr}\ x \mid \mathsf{Ap}\ t_1\ t_2 \mid \mathsf{Lm}\ x\ t$, but interpreted *coinductively*, i.e., allowing an infinite number of constructors. For example, $\ldots\ (\mathsf{Ap}\ (\mathsf{Lm}\ x_n\ (\ldots\ (\mathsf{Ap}\ (\mathsf{Lm}\ x_1\ (\mathsf{Vr}\ x_0))\ (\mathsf{Vr}\ x_1))\ \ldots))\ (\mathsf{Vr}\ x_n))\ \ldots$ is an iterm, infinitely alternating abstractions and applications. Similarly to terms, iterms are equated modulo α.

In more detail, the above definition means: One first defines the set PTr_∞ of *pre-iterms* to be (co)freely generated by the grammar $p ::= \mathsf{PVr}\ x \mid \mathsf{PAp}\ p_1\ p_2 \mid \mathsf{PLm}\ x\ p$ under the coinductive interpretation, i.e., under the assumption that constructors can be applied infinitely. Thus, PTr_∞ is a standard coinductive datatype, given by the final coalgebra of the functor on sets taking, on objects, any set A to $\mathsf{Var} + A \times A + \mathsf{Var} \times A$ (and operating on morphisms as expected; Popescu [2023b, App. E.1] gives full details). Then one defines the α-equivalence relation $\equiv : \mathsf{PTr}_\infty \to \mathsf{PTr}_\infty \to \mathsf{Bool}$ coinductively, proves that it is an equivalence, and defines Tr_∞ by quotienting PTr_∞ to it, i.e., takes $\mathsf{Tr}_\infty = \mathsf{PTr}_\infty/\equiv$. Finally, one proves that the pre-iterm constructors are compatible with $\equiv$,

which allows to define the constructors on iterms: $\mathsf{Vr} : \mathsf{Var} \to \mathsf{Tr}_\infty$, $\mathsf{Ap} : \mathsf{Tr}_\infty \to \mathsf{Tr}_\infty \to \mathsf{Tr}_\infty$ and $\mathsf{Lm} : \mathsf{Var} \to \mathsf{Tr}_\infty \to \mathsf{Tr}_\infty$. We will focus on iterms, forgetting about pre-iterms.

The iterms have been studied in the context of λ-calculus denotational semantics, e.g., the Böhm, Lévy-Longo and Berarducci trees of a λ-term [Barendregt 1985]. A bottom element $\bot$ is often included in the iterm grammar, but we omit it here since it would be entirely passive in our results.

We also consider the usual operators (just like in the inductive case), namely (capture-avoiding) substitution $\_[\_\ /\_] : \mathsf{Tr}_\infty \to \mathsf{Tr}_\infty \to \mathsf{Var} \to \mathsf{Tr}_\infty$, (capture-avoiding) renaming $\_[\_\ /\_] : \mathsf{Tr}_\infty \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Tr}_\infty$, swapping $\_[\_ \wedge \_] : \mathsf{Tr}_\infty \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Tr}_\infty$, permutation $\_[\_] : \mathsf{Tr}_\infty \to \mathsf{Perm} \to \mathsf{Tr}_\infty$, free-variables $\mathsf{FV} : \mathsf{Tr}_\infty \to \mathcal{P}(\mathsf{Var})$, and freshness $\_\#\_ : \mathsf{Var} \to \mathsf{Tr}_\infty \to \mathsf{Bool}$.

Finally, for any set A, let $\mathcal{P}_{\neq\emptyset}(A)$ denote the set of nonempty subsets of A. We consider the iterm *destructor*, $\mathsf{Dest} : \mathsf{Tr}_\infty \to \mathsf{Var} + \mathsf{Tr}_\infty \times \mathsf{Tr}_\infty + \mathcal{P}_{\neq\emptyset}(\mathsf{Var} \times \mathsf{Tr}_\infty)$, defined as follows, where we write V, A and L for the three injections into the sum type $\mathsf{S} = \mathsf{Var} + \mathsf{Tr}_\infty \times \mathsf{Tr}_\infty + \mathcal{P}_{\neq\emptyset}(\mathsf{Var} \times \mathsf{Tr}_\infty)$ (so that $\mathsf{V} : \mathsf{Var} \to \mathsf{S}$, $\mathsf{A} : \mathsf{Tr}_\infty \times \mathsf{Tr}_\infty \to \mathsf{S}$ and $\mathsf{L} : \mathcal{P}_{\neq\emptyset}(\mathsf{Var} \times \mathsf{Tr}_\infty) \to \mathsf{S}$):

$$\mathsf{Dest}\ t = \begin{cases} \mathsf{V}\ x, & \text{if } t = \mathsf{Vr}\ x \\ \mathsf{A}\ (t_1, t_2), & \text{if } t = \mathsf{Ap}\ t_1\ t_2 \\ \mathsf{L}\ \{(x, t') \mid t = \mathsf{Lm}\ x\ t'\}, & \text{otherwise (i.e., if } t \text{ is a Lm-abstraction)} \end{cases}$$

Dest is the dual of the constructors, peeling off the last constructor from an iterm and returning its arguments.[1] It is similar to the destructors for standard datatypes, except that on Lm-abstractions it is nondeterministic. This is because the Lm constructor is not injective and therefore an iterm t could have been built in different ways using Lm. Dest considers all these ways, i.e., returns the set of all pairs (x, t') such that t has the form $\mathsf{Lm}\ x\ t'$. We thus have: $t = \mathsf{Lm}\ x\ t' \iff \exists K.\ \mathsf{Dest}\ t = \mathsf{L}\ K$ and $(x, t') \in K$. For iterms (and for terms too, where the destructor is defined in the same way), destructor and constructors are two faces of the same coin. But since the models for corecursion will have to emulate the destructor, we will look at destructor-based (re)formulations of iterm properties.

Of the basic properties of terms listed in Fig. 2, all except for the last group (the nominal-logic specific properties) also hold for iterms, so we will consider some of them in the context of iterms as well. The properties in this last group are tied to the finiteness of a term's free variables; for them to become true for iterms, we must replace "(in)finite" with "(un)countable".

Moreover, Fig. 6 collects destructor-based iterm counterparts of some term properties from Fig. 2. Often, these are just (equivalent) destructor-based reformulations of the constructor-based properties. For example, this is the case of SwVr_∞, SwAp_∞, SwLm_∞ versus SwVr, SwAp, SwLm.

However, sometimes we reformulate not the original property from Fig. 2, but a converse (or "almost converse") of it. For example, the converse of SwCg from Fig. 2 is: $\mathsf{Lm}\ x_1\ t_1 = \mathsf{Lm}\ x_2\ t_2$ implies that there exists z such that $z \notin \{x_1, x_2\}$, $z \# t_1, t_2$, and $t_1[z \wedge x_1] = t_2[z \wedge x_2]$. This converse does hold for terms, and for iterms as well. However, we prefer to consider a weaker version of it: $\mathsf{Lm}\ x_1\ t_1 = \mathsf{Lm}\ x_2\ t_2$ implies that there exists z such that ($z = x_1$ or $z \# t_1$), ($z = x_2$ or $z \# t_2$), and $t_1[z \wedge x_1] = t_2[z \wedge x_2]$. The latter, reformulated using destructor notation, is exactly SwCg_∞ from Fig. 6. The reason why we prefer a weaker version (here due to a weaker conclusion) is the same as why we preferred a weaker version of SwCg in the inductive case (there, due to a stronger hypothesis): because, to make the (co)recursors as expressive as possible, we want the models to have axioms as weak as possible. Sometimes we include in Fig. 6 two different destructor-based counterparts of a constructor-based property, e.g., RnBvr_∞ and RnBvr'_∞ for RnBvr.

Save for the finite vs. countable nuance in the last group, all properties in Figs. 2 and 6 hold for both terms and iterms. Their selection becomes relevant when regarding them as properties of models. The duality between the Fig. 2 and Fig. 6 properties, which informs the naming of the latter, is neither perfect nor fully systematic. But this naming will allow us to draw parallels.

[1] See page 26 for a discussion of alternative types for the destructor and destructor-like operators.

SwVr_∞	if Dest $t = \mathsf{V}\ x$ then $\mathrm{Dest}(t[z_1 \wedge z_2]) = \mathsf{V}(x[z_1 \wedge z_2])$
SwAp_∞	if Dest $t = \mathsf{A}(t_1, t_2)$ then $\mathrm{Dest}(t[z_1 \wedge z_2]) = \mathsf{A}(t_1[z_1 \wedge z_2], t_2[z_1 \wedge z_2])$
SwLm_∞	if Dest $t = \mathsf{L}\ K$ then there exists K' such that Dest $(t[z_1 \wedge z_2]) = \mathsf{L}\ K'$ and $((x[z_1 \wedge z_2], t'[z_1 \wedge z_2]) \in K'$ for all $(x, t') \in K)$
SwCg_∞	if Dest $t = \mathsf{L}\ K$ and $\{(x_1, t_1), (x_2, t_2)\} \subseteq K$ then there exists z such that $(z = x_1$ or $z \# t_1)$, $(z = x_2$ or $z \# t_2)$, and $t_1[z \wedge x_1] = t_2[z \wedge x_2]$
SwBvr_∞	if Dest $s = \mathsf{L}\ K$ and $\{(x, t), (x', t')\} \subseteq K$ then $(x' = x$ or $x' \# t)$ and $t' = t[x' \wedge x]$
$\mathsf{SwBvr}_{\infty,2}$	same as SwBvr_∞ but with $x' \notin \mathrm{FV}\ t$ instead of $x' \# t$

RnVr_∞	if Dest $t = \mathsf{V}\ x$ then $\mathrm{Dest}(t[y/z]) = \mathsf{V}(x[y/z])$
RnAp_∞	if Dest $t = \mathsf{A}(t_1, t_2)$ then $\mathrm{Dest}(t[y/z]) = \mathsf{A}(t_1[y/z], t_2[y/z])$
$\mathsf{RnLm}_{1,\infty}$	if Dest $t = \mathsf{L}\ K$ then there exists K' s.t. Dest $(t[y/z]) = \mathsf{L}\ K'$ and $((x, t'[y/z]) \in K'$ for all $(x, t') \in K$ s.t. $x \notin \{y, z\})$
$\mathsf{RnLm}_{2,\infty}$	if Dest $t = \mathsf{L}\ K$ and $(x, t') \in K$ then $t[x/z] = t$
RnCg_∞	if Dest $s = \mathsf{L}\ K$ and $\{(x_1, t_1), (x_2, t_2)\} \subseteq K$ then there exists z such that $(z = x_1$ or $z \# t_1)$, $(z = x_2$ or $z \# t_2)$, and $t_1[z/x_1] = t_2[z/x_2]$
RnBvr_∞	if Dest $s = \mathsf{L}\ K$ and $\{(x, t), (x', t')\} \subseteq K$ then $(x' = x$ or $x' \# t)$ and $t' = t[x'/x]$
RnBvr'_∞	if Dest $s = \mathsf{L}\ K$, $(x, t) \in K$ and $x' \# t$ then $(x', t[x'/x]) \in K$

FrVr_∞	if Dest $t = \mathsf{V}\ x$ and $z \# t$ then $z \neq x$
FrAp_∞	if Dest $t = \mathsf{A}(t_1, t_2)$ and $z \# t$ then $z \# t_1$ and $z \# t_2$
FrLm_∞	if Dest $t = \mathsf{L}\ K$, $(x, t') \in K$ and $z \# t$ then $z = x$ or $z \# t'$

FvVr_∞	if Dest $t = \mathsf{V}\ x$ then $x \in \mathrm{FV}(t)$
FvAp_∞	if Dest $t = \mathsf{A}(t_1, t_2)$ then $\mathrm{FV}\ t_1 \cup \mathrm{FV}\ t_2 \subseteq \mathrm{FV}\ t$
FvLm_∞	if Dest $t = \mathsf{L}\ K$ and $(x, t') \in K$ then $\mathrm{FV}\ t' \setminus \{x\} \subseteq \mathrm{FV}\ t$

PmVr_∞	if Dest $t = \mathsf{V}\ x$ then $\mathrm{Dest}(t[\sigma]) = \mathsf{V}(x[\sigma])$
PmAp_∞	if Dest $t = \mathsf{A}(t_1, t_2)$ then $\mathrm{Dest}(t[\sigma]) = \mathsf{A}(t_1[\sigma], t_2[\sigma])$
PmLm_∞	if Dest $t = \mathsf{L}\ K$ then there exists K' such that Dest $(t[\sigma]) = \mathsf{L}\ K'$ and $((x[\sigma], t'[\sigma]) \in K'$ for all $(x, t') \in K)$
PmBvr_∞	if Dest $s = \mathsf{L}\ K$ and $\{(x, t), (x', t')\} \subseteq K$ then $(x' = x$ or $x' \notin \mathrm{FV}\ t)$ and $t' = t[x' \leftrightarrow x]$
PmBvr'_∞	if Dest $s = \mathsf{L}\ K$, $(x, t) \in K$ and $x' \# t$ then $(x', t[x \leftrightarrow x]) \in K$

SbVr_∞	if Dest $t = \mathsf{V}\ x$ then $\mathrm{Dest}(t[s/z]) =$ (if $x = z$ then Dest s else $\mathsf{V}\ x$)
SbAp_∞	if Dest $t = \mathsf{A}(t_1, t_2)$ then $\mathrm{Dest}(t[s/z]) = \mathsf{A}(t_1[s/z], t_2[s/z])$
SbLm_∞	if Dest $t = \mathsf{L}\ K$ then there exists K' such that Dest $(t[s/z]) = \mathsf{L}\ K'$ and $((x, t'[s/z]) \in K'$ for all $(x, t') \in K$ such that $x \neq z$ and $x \# s)$
SbBvr_∞	if Dest $s = \mathsf{L}\ K$ and $\{(x, t), (x', t')\} \subseteq K$ then $(x' = x$ or $x' \# t)$ and $t' = t[(\mathsf{Vr}\ x')/x]$
SbBvr'_∞	if Dest $s = \mathsf{L}\ K$, $(x, t) \in K$ and $x' \# t$ then $(x', t[(\mathsf{Vr}\ x')/x]) \in K$

FSupFv_∞	FV t is countable
FvDPm_∞	$\mathrm{FV}\ t = \{x \in \mathrm{Var} \mid \{y \mid t[x \leftrightarrow y] \neq t\}$ is uncountable$\}$
FvDSw_∞	$\mathrm{FV}\ t = \{x \in \mathrm{Var} \mid \{y \mid t[x \wedge y] \neq t\}$ is uncountable$\}$
FSupFr_∞	$\{x.\ \neg\ x \# t\}$ is countable
FrDSw_∞	$x \# t$ if and only if $\{y \mid t[x \wedge y] \neq t\}$ is countable
FrDRn_∞	$x \# t$ if and only if $\{y \mid t[y/x] \neq t\}$ is countable

Fig. 6. Corecursion-relevant properties of iterms. We only list properties that are counterparts of those from Fig. 2 involving constructors and finiteness conditions. As for the others, namely the algebraic properties of the operators, their formulation does not change, so we will use the same notation. For example, SwId from Fig. 2 denotes a property that makes sense not only for terms but also for iterms (and will, in due course, for our corecursor models too).

5.2 Epi-Corecursors

We introduce abstract epi-corecursors as a natural dual of epi-recursors. The idea is the same: A definition of a morphism in a base category is underpinned by adding more structure coming from an extended category. The difference is that the base object is now not the source, but the target of the to-be-defined morphism, and the underpinning occurs not via initiality but via finality.

Def 17. An *epi-corecursor* is a tuple $cr = (\underline{\mathcal{B}}, T, \underline{\mathcal{C}}, J, R)$ where:

- $\underline{\mathcal{B}}$ is a category called *the base category*
- T is an object in $\underline{\mathcal{B}}$ called *the base object*
- $\underline{\mathcal{C}}$ is a category called *the extended category*
- J is a final object in $\underline{\mathcal{C}}$
- $R : \underline{\mathcal{C}} \to \underline{\mathcal{B}}$ is a functor such that $R\ J = T$

Just like for epi-recursors, in typical epi-corecursor examples $\underline{\mathcal{C}}$ and $\underline{\mathcal{B}}$ will be categories of models, with the models in $\underline{\mathcal{C}}$ having more structure than those in $\underline{\mathcal{B}}$, and R will be a structure-forgetting functor. To define a morphism $g : B \to T$ in $\underline{\mathcal{B}}$ (where B is some object in $\underline{\mathcal{B}}$) using an epi-corecursor $cr = (\underline{\mathcal{B}}, T, \underline{\mathcal{C}}, J, R)$, we (1) extend B to an object C in $\underline{\mathcal{C}}$ (with $R\ C = B$) yielding a morphism $!_{C,J} : C \to J$ in $\underline{\mathcal{C}}$ from the finality of J, then (2) take g to be $R\ !_{C,J}$, the restriction of $!_{C,J}$ to $\underline{\mathcal{B}}$. Thus, we call a morphism $g : B \to T$ *definable by the epi-corecursor cr* if $g = R\ !_{C,J}$ for some extension C of B.

5.3 A Hierarchy of Nominal Corecursors

To discuss concrete nominal corecursors, we adapt §3.2's signatures and models used for nominal recursors. We use the same notions except that we replace the constructor symbols vr, ap, lm with a destructor symbol dest, interpreted accordingly. All signatures Σ now extend not the constructor signature $\Sigma_{\mathsf{ctor}} = \{\mathsf{vr}, \mathsf{ap}, \mathsf{lm}\}$, but the destructor signature $\Sigma_{\mathsf{dtor}} = \{\mathsf{dest}\}$. A Σ-model $\mathcal{M}$ has a carrier set M, interprets the signature's non-destructor symbols as in §3.2, and dest as an operation $\mathsf{Dest}^{\mathcal{M}} : M \to \mathsf{Var} + M \times M + \mathcal{P}_{\neq\emptyset}(\mathsf{Var} \times M)$. The iterm Σ-model $\mathcal{T}r_\infty(\Sigma)$ is the Σ-model whose carrier set is Tr_∞ and whose operations and relations are the standard ones for iterms (see §5.1).

The notion of morphism of Σ-models $g : \mathcal{M} \to \mathcal{M}'$ is defined like in §3.2, but replacing commutation with the constructors by sub-commutation with the destructor: $(1_{\mathsf{Var}} + g \times g + \mathsf{image}(1_{\mathsf{Var}} \times g))\ (\mathsf{Dest}^{\mathcal{M}} m) \sqsubseteq \mathsf{Dest}^{\mathcal{M}'}(g\ m)$ for all m in the carrier set M. The above relation $\sqsubseteq$ on $\mathsf{Var} + M' \times M' + \mathcal{P}_{\neq\emptyset}(\mathsf{Var} \times M')$ is defined by taking $u \sqsubseteq v$ to mean that: either $u = \mathsf{V}\ x = v$ for some x; or $u = \mathsf{A}(m'_1, m'_2) = v$ for some m'_1, m'_2; or $u = \mathsf{L}\ K$, $v = \mathsf{L}\ K'$ and $K \subseteq K'$ for some K, K'. Thus, the *sub*-commutation shows in the abstraction case (which is nondeterministic), where we allow inclusion instead of equality. To see why sub-commutation is the natural condition here, note that for a morphism that targets iterms, $g : \mathcal{M} \to \mathcal{T}r_\infty(\Sigma)$, it is equivalent to the conjunction of the following three conditions: (1) $\mathsf{Dest}^{\mathcal{M}}\ m = \mathsf{V}\ x$ implies $g\ m = \mathsf{Vr}\ x$; (2) $\mathsf{Dest}^{\mathcal{M}}\ m = \mathsf{A}(m_1, m_2)$ implies $g\ m = \mathsf{Ap}\ (g\ m_1)\ (g\ m_2)$; (3) $\mathsf{Dest}^{\mathcal{M}}\ m = \mathsf{L}\ K$ and $(x, m') \in K$ implies $g\ m = \mathsf{Lm}\ x\ (g\ m')$.

Our nominal (epi-)corecursors will underpin corecursive definitions having $\mathcal{T}r_\infty(\Sigma_{\mathsf{dtor}})$ as target model by considering extensions of Σ_{dtor} to larger signatures Σ, along with certain axiomatizations of Σ-models given by subsets of the properties in Fig. 6 (interpreted not on iterms, but on Σ-models).

Previous work [Blanchette et al. 2019; Kurz et al. 2012] discovered corecursive counterparts of two nominal recursors. Next we show that this is a quite pervasive phenomenon:

Thm 18. Consider the eight choices, for $i \in \{1, 2, 3, 5, 6, 7, 8, 9\}$, of tuples $cr_i = (\underline{\mathcal{B}}, T, \underline{\mathcal{C}}_i, J_i, R_i)$ given by the sets of properties Props_i shown in Fig. 7. Namely (analogously to what we assumed in Thm. 9), we assume that Σ_i consists of the operation and relation symbols occurring in Props_i, and:

- $\underline{\mathcal{B}}$ is the category of Σ_{dtor}-models and $T = \mathcal{T}r_\infty(\Sigma_{\mathsf{dtor}})$
- $\underline{\mathcal{C}}_i$ is the category of $(\Sigma_i, \mathsf{Props}_i)$-models and J_i is $\mathcal{T}r_\infty(\Sigma_i)$
- $R_i : \underline{\mathcal{C}}_i \to \underline{\mathcal{B}}_i$ is the forgetful functor sending $(\Sigma_i, \mathsf{Props}_i)$-models to their underlying Σ_{dtor}-models

Then cr_i is an epi-corecursor. In particular, $\mathcal{T}r_\infty(\Sigma_i)$ is the final $(\Sigma_i, \mathsf{Props}_i)$-model.

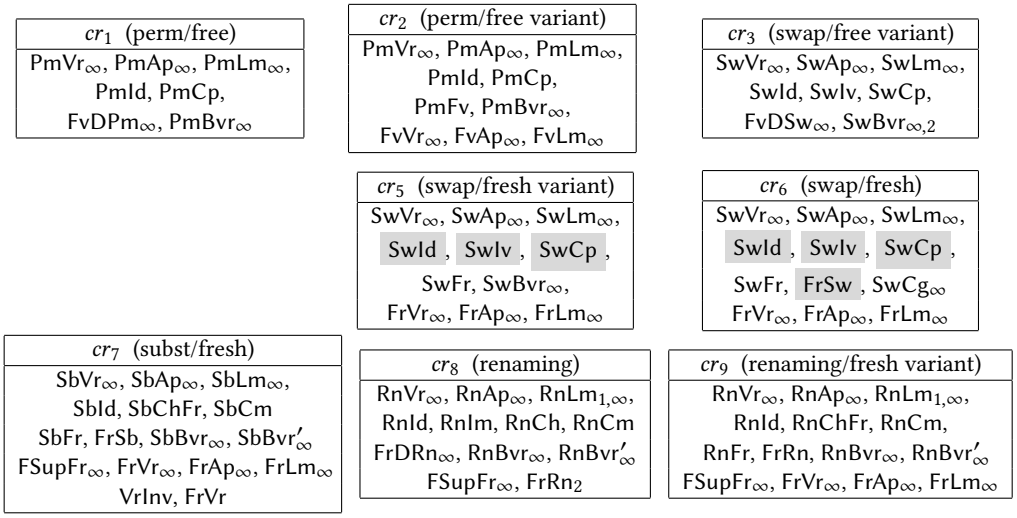

Fig. 7. Sets of properties underlying different nominal corecursors. The highlighted properties are ones that turned out to be redundant in the analogous nominal recursor, but must be added back for the corecursor.

Next we unpack Thm. 18's statements of epi-corecursion principles, exploring the connections with Thm. 9's nominal epi-recursors. We used for the corecursors the same names as for the recursors to which they roughly correspond—although, as we will discuss, a corecursor will often "inherit" axioms from two different recursors. (We do not have a cr_4 corecursor because the axioms specific to r_4 were mixed into cr_5 and cr_6; either of cr_5 and cr_6 could have alternatively been named "cr_4".)

cr_1 and cr_3 are corecursors in the style of nominal logic. Like their recursor counterparts r_1 and r_3, they have the free-variable (support) operator completely determined from permutation (via FvDPm$_\infty$), or alternatively swapping (via FvDSw$_\infty$). However, these corecursors are not strictly speaking nominal-logic based, because this determination of free-variables involves not finiteness, but countability. Another difference between cr_1 / cr_3 and r_1 / r_3 is that the freshness condition for binders FCB (or anything analogous to it) is no longer needed; but instead we need the (corecursive counterpart of) the bound-variable renaming axiom which was specific to the more expressive recursor r_5—in permutation or swapping form (PmBvr$_\infty$ or SwBvr$_{\infty,2}$). Thus, when switching from recursion to corecursion, the nominal-logic style definitional principles trade FCB for PmBvr$_\infty$ or SwBvr$_{\infty,2}$; they are the only ones *not* to become axiomatically heavier during this switch.

The cr_2 corecursor requires both the algebraic properties of permutation and freshness specific to r_2 (PmId, PmCp and PmFv) and the bound-variable renaming property specific to r_5 (converted from swapping to permutation form, PmBvr$_\infty$). The situation is similar for cr_5, the swapping-based counterpart of cr_2, which gets axioms from both r_4 (with freeness converted to freshness) and r_5. All these are in sharp contrast to the recursion case, where, at the recursor r_5, bound-variable renaming (SwBvr) was the only axiom needed (in addition to the "unavoidable" ones describing the interaction of constructors with the other operators). Similarly to cr_5 which "descends" from r_4 and r_5, cr_6 "descends" from r_4 and r_6. Unlike in the recursive case where r_4 did not need SwCp and turned out not to need SwId and SwIv either, here all three axioms, SwId, SwIv and SwCp, are actually needed by its corecursor "descendants" cr_5 and cr_6. Additionally cr_6 requires FvSw, another axiom we had discovered to be redundant for r_4. Thus, for the principles discussed in this paragraph, the axiomatizations become heavier when switching from recursion to corecursion, because: (1) axioms from different recursors now need to be joined, and (2) previous axioms that were seen to be redundant for recursors must be added back to their corecursor counterparts.

As for the substitution- and renaming-based principles cr_7, cr_8 and cr_9, their axiomatizations also become heavier in a similar way, in that both algebraic axioms (e.g., RnId, RnIm, RnCh, RnCm) and bound-variable renaming axioms must be present. But their axiomatizations are even heavier, because they feature (1) two versions of the bound-variable renaming axioms (e.g., RnBvr_∞ and RnBvr'_∞ as opposed to just RnBvr_∞) as well as (2) countable support (FSupFr_∞). Roughly speaking, these additional axioms are needed to make corecursion go through (i.e., establish finality of the iterm model) because, substitution/renaming not commuting unconditionally with abstractions, stronger bound-variable avoidance facilities must be supplied by an (arbitrary) model; this was not a problem for recursors, where fresh induction on (concrete) terms could handle that elegantly.

Specific to the substitution corecursor cr_7 is that it features, for the variable case, not only the destructor freshness axiom FrVr_∞, but also its *constructor* counterpart FrVr, and the implicit requirement that the signature Σ_7 contains the variable-constructor symbol vr. So a Σ_7-model $\mathcal{M}$ has, in addition to the destructor $\mathsf{Dest}^{\mathcal{M}}$, a variable-constructor-like operator $\mathsf{Vr}^{\mathcal{M}} : \mathsf{Var} \to \mathcal{M}$; the two are required to act as mutual inverses by the following axiom VrInv (which, due its hybrid nature, fits neither Fig. 2 nor Fig. 6): $\mathsf{Dest}\ t = \mathsf{V}\ x$ if and only if $t = \mathsf{Vr}\ x$. This monad-like variable-injection setting is needed to accommodate the substitution of arbitrary elements $m \in M$ for variables x.

Connection with previous corecursors. Thm. 18 recovers, and slightly improves on, the two existing nominal corecursors from the literature we are aware of: that developed by Kurz et al. [2012] for λ-terms and extended by Kurz et al. [2013] to functors on nominal sets, and that developed by Blanchette et al. [2019] in a functorial framework covering complex binders. Next we discuss these corecursors' instantiations to the syntax of λ-calculus. The Blanchette et al. corecursor corresponds to cr_2 almost exactly, just that it assumes FvPm which is not needed. (See Popescu [2023b, App. F.2].)

Designed for nominal logic, the Kurz et al. corecursor assumes finite support, and targets not the entire Tr_∞ but the subset $\mathsf{Tr}'_\infty \subseteq \mathsf{Tr}_\infty$ of finitely supported iterms. Their corecursor can be obtained from our cr_1 by noting that, if we assume the source model $\mathcal{M}$ to satisfy finite support (FSupFv), then the image of the unique morphism $g : \mathcal{M} \to \mathcal{T}r_\infty(\Sigma_1)$ guaranteed by cr_1 is included in Tr'_∞ (thanks to g's preservation of free variables). So we obtain a unique morphism from $\mathcal{M}$ to the submodel of $\mathcal{T}r_\infty(\Sigma_1)$ with carrier set Tr'_∞, i.e., the term model of Kurz et al. The above summary ignores one technicality: The Kurz et al. destructor does not have type $M \to \mathsf{Var} + M \times M + \mathcal{P}_{\neq\emptyset}(\mathsf{Var} \times M)$ like ours, but $M \to \mathsf{Var} + M \times M + [\mathsf{Var}]M$, where $[\mathsf{Var}]M$ is the nominal set of abstractions, obtained by quotienting $\mathsf{Var} \times M$ to an α-like equivalence relation $\sim$ defined by $(x, m) \sim (x', m')$ iff $m[z \leftrightarrow x]^{\mathcal{M}} = m[z \leftrightarrow x']^{\mathcal{M}}$ for some fresh z. Since $[\mathsf{Var}]M$ consists of $\sim$-equivalence classes, we have $[\mathsf{Var}]M \subseteq \mathcal{P}_{\neq\emptyset}(\mathsf{Var} \times M)$, so the only difference is that our destructor has a less constrained codomain. But our cr_1 axiom PmBvr_∞ constrains the elements of $\mathcal{P}_{\neq\emptyset}(\mathsf{Var} \times M)$ from the image of the destructor to contain mutually $\sim$-equivalent items. If we also added PmBvr'_∞ to the axiomatization of cr_1, we would further constrain these to be entire $\sim$-equivalence classes, obtaining exactly the Kurz et al. models. Hence, due to its models being less constrained, cr_1 is (slightly) more expressive than the Kurz et al. corecursor.

A note on nominal abstractions. The above recalled abstractions are a standard concept in nominal logic [Gabbay and Pitts 1999], and using abstractions as primitives is a valid alternative when introducing nominal recursors and corecursors. For the recursors, the Lm-constructor in models would have type $[\mathsf{Var}]M \to M$ rather than $\mathsf{Var} \to M \to M$. However, like the authors of the nominal recursors reviewed in §2.2, we too favor the abstraction-free (hence quotient-free) (co)recursors, and this is for two reasons. First, they are likely easier to deploy: During a recursive definition, it seems inconvenient for the user to have to provide an operator in $[\mathsf{Var}]M \to M$, which usually requires making a choice and showing that the choice is immaterial; providing instead a "free" operator in $\mathsf{Var} \to M \to M$ and verifying an additional axiom (such as SwBvr) seems more manageable. Second, they can be more expressive than their abstraction-based alternatives. For example, most of the recursors

in Thm. 9 do not require swapping/permutation to have the algebraic properties needed for $\sim$ to be an equivalence, so quotienting is not an option unless we strengthen the model axiomatization, thus placing a higher proof burden on the user. Admittedly, these advantages are less consequential when talking about corecursors, where the relevant algebraic properties are required across the board.

Comparing expressiveness. We use a strength relation that is similar to that from our "head-to-head" comparison of epi-recursors (in §4.1): Given epi-corecursors $cr = (\underline{\mathcal{B}}, T, \underline{C}, J, R)$ and $cr' = (\underline{\mathcal{B}}, T, \underline{C}', J', R')$, we call cr' *stronger than* cr, written $cr' \geq cr$, if cr' can define everything that cr can, in that: for all objects B in $\underline{\mathcal{B}}$ and $g : B \to T$, g definable by cr implies g definable by cr'. Again, we write $cr \equiv cr'$ to state that cr and cr' have equal strengths, i.e., both $cr' \geq cr$ and $cr \geq cr'$ hold.

Thm 19. The epi-corecursors from Thm. 18 (and Fig. 7) compare as follows w.r.t. expressiveness:
$cr_2 \equiv cr_5 \geq cr_6 \geq cr_3 \equiv cr_1 \geq cr_8$ and $cr_5 \geq cr_9 \geq cr_7, cr_8$.

Let us discuss this hierarchy in connection with the recursor hierarchy from Thm. 12:

Permutation versus swapping. Recall that, in the recursor hierarchy, choosing between permutation and swapping was consequential to expressiveness as soon as we no longer assumed the tight coupling between freeness/freshness and swapping/permutation; namely, for the tight-coupling recursors r_1 and r_3 we had $r_1 \equiv r_3$, but for the for loose-coupling recursors r_2 and r_4 we only had $r_4 \geq r_2$. But on corecursors this nuance disappears: Swapping is now as expressive as permutation in both the tight-coupling ($cr_1 \equiv cr_3$) and loose-coupling ($cr_2 \equiv cr_5$) cases. This is because for swapping-based corecursors we cannot dispense with the algebraic axioms SwId, SwIv and SwCp, which are sufficient to ensure the extension of swapping to a (well-behaved) permutation operator.

Congruence versus bound-variable renaming. Recall that, for recursors, the congruence axiom SwCg led to higher expressiveness than the bound-variable renaming axiom SwBvr, yielding $r_6 \geq r_5$. And this was because (in the presence of other mild axioms) SwBvr implies SwCg. The same is true here for corecursors, in that SwBvr_∞ implies SwCg_∞. However, in the presence of the other cr_5 axioms, SwBvr_∞ is sufficient for proving a corecursion principle; whereas SwCg_∞ is not, unless we add the additional axiom FrSw (which is not needed by cr_5). And if we assume FrSw then SwCg_∞ also implies SwBvr_∞. In short, congruence-based corecursion requires FrSw, and as such is less expressive than bound-variable renaming-based corecursion, meaning that the hierarchy gets shifted, with $cr_5 \geq cr_6$.

Symmetric versus asymmetric operators, second round. Recall that, in the strict "head-to-head" comparison relation, recursors based on symmetric operators (swapping and permutation) were incomparable to those based on asymmetric ones (renaming and substitution), but only a laxer comparison deemed the symmetric ones more expressive. But in the case of corecursors, the symmetric ones emerge as more expressive already in a head-to-head comparison. This is not too surprising if we recall the reason why symmetric-operator recursors eventually emerged as more expressive: because, if the model has finite support (which in the laxer criterion was possible by taking the minimal submodel), then swapping becomes definable from renaming. Here, our asymmetric-operator based models already have countable support (which, as discussed, seems necessary for corecursion), hence can also define swapping from renaming similarly to how this is done in the finite-support case.

Summary. Nominal corecursors can be construed and compared as epi-corecursors, following a similar methodology to that for nominal recursors. A corecursor axiomatization corresponds to one or two recursor axiomatizations via identical and quasi-dual axioms. The corecursor axiomatizations are heavier. We have a corecursor hierarchy that partly matches the strict-relation ($\geq$) recursor hierarchy but is more fine-grained, in particular it already subsumes asymmetric-operator principles to the symmetric-operator ones without the need for a laxer comparison relation (in the style of $\gtrsim$).

6 MECHANIZED RESULTS

We have mechanized in Isabelle/HOL the recursion theorem (Thm. 9), the two recursor comparison theorems (Thms. 12 and 15), the two negative (strictness) results on recursor comparison (Prop. 16), the corecursion theorem (Thm. 18), and the corecursor comparison theorem (Thm. 19). What we have *not* mechanized are the abstract criteria for comparing epi-recursors, namely Props. 11 and 14. In our mechanized results, rather than invoking these criteria, we have inlined their content on a need basis.

The mechanization is available as an archive [Popescu 2023a], and is extensively documented in Popescu [2023b, App. J]. It uses Isabelle's structuring mechanisms called *locales* [Ballarin 2014; Kammüller et al. 1999] to represent the model axiomatizations, and uses sublocale relationships for the transformations between these axiomatizations that underlie the expressiveness comparisons.

We have also provided a top-level, locale-free reformulation of the mechanized results, which match closely the statements from the paper, and whose inspection does not require knowledge of locales. The end results about recursors, Thms. 9, 12 and 15 and Prop. 16, are mechanized in homonymous Isabelle theories, located in the archive's directory Stripped_Down/LocaleFree_versions:

- Thm. 9 is mechanized in the Isabelle theory Theorem9. That theory contains the definitions of the epi-recursor structure for each of the nine recursors r_i, and proofs that these structures indeed form epi-recursors, e.g., their components are categories, functors etc. The initiality theorems are named init_Ii where i is a number between 1 and 9.
- Thm. 12 is mechanized in the Isabelle theory Theorem12, where the main formal theorems are named ri_ge_rj (formalizing $r_i \geq r_j$) for the relevant choices of i and j.
- Thm. 15 is mechanized in the Isabelle theory Theorem15, where the main formal theorems are named ri_quasi_ge_rj (formalizing $r_i \gtrsim r_j$), again for the relevant choices of i and j.
- Prop. 16 is mechanized in the Isabelle theory Prop16, where the main formal theorems are named not_r1_ge_r2 and not_r2_ge_r4 (formalizing $r_1 \not\geq r_2$ and $r_2 \not\geq r_4$).

And similarly for corecursors, in directory Corecursors/LocaleFree_versions:

- Thm. 18 is mechanized in the Isabelle theory Theorem18, which contains the definitions and proofs for the epi-corecursor structure, including the finality theorems named final_Ji.
- Thm. 19 is mechanized in the Isabelle theory Theorem19, where the main formal theorems are named cr$i$_ge_cr$j$ (formalizing $cr_i \geq cr_j$) for the relevant choices of i and j.

Popescu [2023b, App. J.4] gives more details about the locale-free statements of the results.

7 MORE RELATED WORK

Definitional packages for syntax with bindings. A direct application of our results would be on informing the design of binding-aware definitional packages in proof assistants, in the style of Nominal Isabelle [Urban and Kaliszyk 2012]. In addition to our theoretical results on expressiveness, one should also consider the pragmatic aspects of how lightweight the required structure (operations and relations on the target domain) is and how easy the conditions are to solve. Ideally, in a definitional package implementation one should provide the maximally expressive (co)recursor as the core, but also infer from it (via "borrowing") and make available other (co)recursors which may have pragmatic advantages. For example, the recursors r_3 and r_8 are minimalistic in terms of structure.

(Co)recursors in different paradigms. Binding-aware recursors have also been developed in the other two major paradigms. Scope-safe versions of nameless recursion based on category theory have been studied extensively, e.g., Allais et al. [2017]; Altenkirch and Reus [1999]; Bird and Paterson [1999]; Fiore et al. [1999]; Hofmann [1999]; Kaiser et al. [2018]. A nameless recursor is in principle easier to deploy because the constructors are free; the price is additional index-shifting

overhead [Berghofer and Urban 2007]. Nameless corecursion has been studied by Matthes and Uustalu [2004], building on previous work by Aczel et al. [2003]; Ghani et al. [2003]; Moss [2001].

Hybrid nameless/nominal solutions have also been proposed, notably the locally named [McKinna and Pollack 1999; Pollack et al. 2012] and locally nameless [Aydemir et al. 2008; Charguéraud 2012] representations. Pitts [2023] introduced locally nameless sets, an algebraic axiomatization of syntax under the locally nameless representation, and characterizes the locally nameless recursor [Charguéraud 2012] using initiality in a functor category (similarly to recursors in the nameless setting [Fiore et al. 1999; Hofmann 1999]). He also proved that the category of locally nameless sets is isomorphic to that of finitely supported rensets [Popescu 2023c] and to categories given by other axiomatizations of renaming from the literature [Gabbay and Hofmann 2008; Staton 2007]; this suggests that the expressive power of the locally nameless recursor might be located in the vicinity of r_8 (which is based on rensets). On the way to his results, Pitts gave an alternative axiomatization of finitely supported rensets, using instead of RnCh a simpler (unconditional) axiom, let us call it RnCh': $t[x_2/x_1][x_3/x_2] = t[x_3/x_2][x_3/x_1]$. Replacing RnCh with RnCh' would yield a recursor r_8' such that $r_8 \geq r_8'$ (since RnCh' implies RnCh in the presence of RnIm) and $r_8 \cong r_8'$ (since the converse implication is true for finitely supported rensets, hence for a suitable minimal submodel).

In *strong HOAS*, as implemented in dedicated logical frameworks [Baelde et al. 2014; Pfenning and Schürmann 1999; Pientka 2010], the λ-constructor has type $(\mathsf{Tr} \to \mathsf{Tr}) \to \mathsf{Tr}$. Here, the difficulty with recursion is not the non-freeness of the constructors, but the fact that binding constructors are not recursable in the typical well-foundedness manner. Solutions to this have been designed using modality operators [Schürmann et al. 2001] and contextual types [Ferreira and Pientka 2017]. Recursion mechanisms have also been designed within *weak HOAS* [Despeyroux et al. 1995], where the λ-constructor, having type $(\mathsf{Var} \to \mathsf{Tr}) \to \mathsf{Tr}$, *is* standardly recursable—yielding a free datatype that contains all terms but also additional entities referred to as "exotic terms". Partly due to the exotic terms, this free datatype is not very helpful for recursively defining useful functions on terms. But the situation is significantly improved in a variant called *parametric HOAS (PHOAS)* [Chlipala 2008], which accommodates recursive definitions in the style of the semantic-interpretation pattern (§4.2).

A nominal/HOAS hybrid can be found in Gordon and Melham's characterization of the λ-term datatype [Gordon and Melham 1996], which employs the nameful constructors but features weak-HOAS style recursion over Lm. Norrish [2004] inferred his swap/free recursor r_4 from the Gordon-Melham one. Weak-HOAS recursion also has interesting connections with nameless recursion: In presheaf toposes as in Fiore et al. [1999], Hofmann [1999] and Ambler et al. [2003], the function space $\mathsf{Var} \Rightarrow T$ is isomorphic to the De Bruijn level-shifting transformation applied to T; this effectively equates the weak-HOAS and nameless recursors.

Recursion over non-free datatypes. Some of the discussed nominal recursors operate by characterizing terms as the non-free datatype determined as initial model of an equational theory [Burris and Sankappanavar 1981] or more generally of a Horn theory [Makowsky 1987], employing an infinite number of axioms. In such cases, and ignoring the Barendregt enhancement, nominal recursion becomes a particular case of Horn recursion. (This is not true for the nominal-logic recursor r_1, since FvDPm is not a Horn formula.) Our concept of epi-recursor applies to general Horn recursion as well—provided one identifies a constructor-like subsignature of the given signature, i.e., such that the initial model of the Horn theory has its carrier generated by its operations. In algebraic specifications, this property is called *sufficient completeness* [Guttag and Horning 1978].

The non-free datatypes of sets and bags are degenerate cases of the above, where the constructors form the entire signature. Tannen and Subrahmanyam [1991] and Buneman et al. [1995] study Horn recursors for these datatypes when designing database languages. They prove connections between their axiomatizations that could be captured using our $\geq$ relation between epi-recursors.

DATA AVAILABILITY STATEMENT

The artifact associated with this paper, consisting of the Isabelle mechanization described in §6 (and also more extensively in Popescu [2023b, App. J]) is available as a Zenodo entry [Popescu 2023a].

ACKNOWLEDGMENTS

We thank the paper reviewers and the artifact reviewers for the careful reading of our paper, and for their insightful comments and suggestions, which have led to improvements both in the text and in the documentation of what has been mechanized. We gratefully acknowledge support from the EPSRC grant EP/X015114/1 "Safe and secure COncurrent programming for adVancEd aRchiTectures (COVERT)".

REFERENCES

Andreas Abel, Alberto Momigliano, and Brigitte Pientka. 2017. POPLMark Reloaded. In *Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP) 2017*, Marino Miculan and Florian Rabe (Eds.). https://lfmtp.org/workshops/2017/inc/papers/paper_8_abel.pdf

Peter Aczel, Jirí Adámek, Stefan Milius, and Jiri Velebil. 2003. Infinite trees and completely iterative theories: a coalgebraic view. *Theor. Comput. Sci.* 300, 1-3 (2003), 1–45. https://doi.org/10.1016/S0304-3975(02)00728-4

Guillaume Allais, James Chapman, Conor McBride, and James McKinna. 2017. Type-and-scope safe programs and their proofs. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, Yves Bertot and Viktor Vafeiadis (Eds.). ACM, 195–207. https://doi.org/10.1145/3018610.3018613

Thorsten Altenkirch and Bernhard Reus. 1999. Monadic Presentations of Lambda Terms using Generalized Inductive Types. In *Computer Science Logic (CSL) 1999*, Jörg Flum and Mario Rodríguez-Artalejo (Eds.). LNCS, Vol. 1683. Springer, 453–468. https://doi.org/10.1007/3-540-48168-0_32

S. J. Ambler, Roy L. Crole, and Alberto Momigliano. 2003. A definitional approach to primitive recursion over higher order abstract syntax. In *Eighth ACM SIGPLAN International Conference on Functional Programming, Workshop on Mechanized reasoning about languages with variable binding, MERLIN 2003, Uppsala, Sweden, August 2003*. ACM. https://doi.org/10.1145/976571.976572

Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. 2005. Mechanized Metatheory for the Masses: The PoplMark Challenge. In *Theorem Proving in Higher Order Logics (TPHOLs) 2005*, Joe Hurd and Thomas F. Melham (Eds.). LNCS, Vol. 3603. Springer, 50–65. https://doi.org/10.1007/11541868_4

Brian E. Aydemir, Aaron Bohannon, and Stephanie Weirich. 2007. Nominal Reasoning Techniques in Coq (Extended Abstract). *Electr. Notes Theor. Comput. Sci.* 174, 5 (2007), 69–77. https://doi.org/10.1016/j.entcs.2007.01.028

Brian E. Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. 2008. Engineering Formal Metatheory. In *Principles of Programming Languages (POPL) 2008*, George C. Necula and Philip Wadler (Eds.). ACM, 3–15. https://doi.org/10.1145/1328438.1328443

David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu, and Yuting Wang. 2014. Abella: A System for Reasoning about Relational Specifications. *J. Formalized Reasoning* 7, 2 (2014), 1–89. https://doi.org/10.6092/issn.1972-5787/4650

Clemens Ballarin. 2014. Locales: A Module System for Mathematical Theories. *J. Autom. Reason.* 52, 2 (2014), 123–153. https://doi.org/10.1007/s10817-013-9284-7

Hendrik Pieter Barendregt. 1985. *The lambda calculus - its syntax and semantics*. Studies in logic and the foundations of mathematics, Vol. 103. North-Holland.

Stefan Berghofer and Christian Urban. 2007. A Head-to-Head Comparison of de Bruijn Indices and Names. *Electr. Notes Theor. Comput. Sci.* 174, 5 (2007), 53–67. https://doi.org/10.1016/j.entcs.2007.01.018

Richard S. Bird and Ross Paterson. 1999. De Bruijn Notation as a Nested Datatype. *J. Funct. Program.* 9, 1 (1999), 77–91. https://doi.org/10.1017/S0956796899003366

Jasmin Christian Blanchette, Lorenzo Gheri, Andrei Popescu, and Dmitriy Traytel. 2019. Bindings as bounded natural functors. *Proc. ACM Program. Lang.* 3, POPL (2019), 22:1–22:34. https://doi.org/10.1145/3290335

Peter Buneman, Shamim A. Naqvi, Val Tannen, and Limsoon Wong. 1995. Principles of Programming with Complex Objects and Collection Types. *Theor. Comput. Sci.* 149, 1 (1995), 3–48. https://doi.org/10.1016/0304-3975(95)00024-Q

Stanley Burris and Hanamantagouda P. Sankappanavar. 1981. *A course in universal algebra*. Graduate texts in mathematics, Vol. 78. Springer.

Arthur Charguéraud. 2012. The Locally Nameless Representation. *J. Autom. Reasoning* 49, 3 (2012), 363–408. https://doi.org/10.1007/s10817-011-9225-2

Adam Chlipala. 2008. Parametric Higher-Order Abstract Syntax for Mechanized Semantics. In *International Conference on Functional Programming (ICFP) 2008*, James Hook and Peter Thiemann (Eds.). ACM, 143–156. https://doi.org/10.1145/1411204.1411226

Ernesto Copello, Nora Szasz, and Álvaro Tasistro. 2018. Formalisation in Constructive Type Theory of Barendregt's Variable Convention for Generic Structures with Binders. In *Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP) 2018*, Frédéric Blanqui and Giselle Reis (Eds.). EPTCS, Vol. 274. 11–26. https://doi.org/10.4204/EPTCS.274.2

Joëlle Despeyroux, Amy P. Felty, and André Hirschowitz. 1995. Higher-Order Abstract Syntax in Coq. In *Typed Lambda Calculi and Applications (TLCA) 1995*, Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin (Eds.). LNCS, Vol. 902. Springer, 124–138. https://doi.org/10.1007/BFb0014049

Amy P. Felty and Alberto Momigliano. 2012. Hybrid: A Definitional Two-Level Approach to Reasoning with Higher-Order Abstract Syntax. *J. Autom. Reasoning* 48, 1 (2012), 43–105. https://doi.org/10.1007/s10817-010-9194-x

Amy P. Felty, Alberto Momigliano, and Brigitte Pientka. 2018. Benchmarks for reasoning with syntax trees containing binders and contexts of assumptions. *Math. Struct. Comput. Sci.* 28, 9 (2018), 1507–1540. https://doi.org/10.1017/S0960129517000093

Francisco Ferreira and Brigitte Pientka. 2017. Programs Using Syntax with First-Class Binders. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings (Lecture Notes in Computer Science)*, Hongseok Yang (Ed.), Vol. 10201. Springer, 504–529. https://doi.org/10.1007/978-3-662-54434-1_19

Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. 1999. Abstract Syntax and Variable Binding. In *Logic in Computer Science (LICS) 1999*. IEEE Computer Society, 193–202. https://doi.org/10.1109/LICS.1999.782615

Murdoch Gabbay and Andrew M. Pitts. 1999. A New Approach to Abstract Syntax Involving Binders. In *Logic in Computer Science (LICS) 1999*. IEEE Computer Society, 214–224. https://doi.org/10.1109/LICS.1999.782617

Murdoch James Gabbay and Martin Hofmann. 2008. Nominal Renaming Sets. In *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings (Lecture Notes in Computer Science)*, Iliano Cervesato, Helmut Veith, and Andrei Voronkov (Eds.), Vol. 5330. Springer, 158–173. https://doi.org/10.1007/978-3-540-89439-1_11

Neil Ghani, Christoph Lüth, Federico De Marchi, and John Power. 2003. Dualising Initial Algebras. *Math. Struct. Comput. Sci.* 13, 2 (2003), 349–370. https://doi.org/10.1017/S0960129502003912

Lorenzo Gheri and Andrei Popescu. 2020. A Formalized General Theory of Syntax with Bindings: Extended Version. *J. Autom. Reason.* 64, 4 (2020), 641–675. https://doi.org/10.1007/s10817-019-09522-2

Andrew D. Gordon and Thomas F. Melham. 1996. Five Axioms of Alpha-Conversion. In *Theorem Proving in Higher Order Logics, 9th International Conference, TPHOLs'96, Turku, Finland, August 26-30, 1996, Proceedings (Lecture Notes in Computer Science)*, Joakim von Wright, Jim Grundy, and John Harrison (Eds.), Vol. 1125. Springer, 173–190. https://doi.org/10.1007/BFb0105404

John V. Guttag and James J. Horning. 1978. The Algebraic Specification of Abstract Data Types. *Acta Informatica* 10 (1978), 27–52. https://doi.org/10.1007/BF00260922

Martin Hofmann. 1999. Semantical Analysis of Higher-Order Abstract Syntax. In *Logic in Computer Science (LICS) 1999*. IEEE Computer Society, 204–213. https://doi.org/10.1109/LICS.1999.782616

Jonas Kaiser, Steven Schäfer, and Kathrin Stark. 2018. Binder aware recursion over well-scoped de Bruijn syntax. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, June Andronick and Amy P. Felty (Eds.). ACM, 293–306. https://doi.org/10.1145/3167098

Florian Kammüller, Markus Wenzel, and Lawrence C. Paulson. 1999. Locales - A Sectioning Concept for Isabelle. In *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings (Lecture Notes in Computer Science)*, Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin-Mohring, and Laurent Théry (Eds.), Vol. 1690. Springer, 149–166. https://doi.org/10.1007/3-540-48256-3_11

Alexander Kurz, Daniela Petrişan, Paula Severi, and Fer-Jan de Vries. 2012. An Alpha-Corecursion Principle for the Infinitary Lambda Calculus. In *Coalgebraic Methods in Computer Science (CMCS) 2012*, Dirk Pattinson and Lutz Schröder (Eds.). LNCS, Vol. 7399. Springer, 130–149. https://doi.org/10.1007/978-3-642-32784-1_8

Alexander Kurz, Daniela Petrişan, Paula Severi, and Fer-Jan de Vries. 2013. Nominal Coalgebraic Data Types with Applications to Lambda Calculus. *Logical Methods in Computer Science* 9, 4 (2013). https://doi.org/10.2168/LMCS-9(4:20)2013

Johann A. Makowsky. 1987. Why Horn Formulas Matter in Computer Science: Initial Structures and Generic Examples. *J. Comput. Syst. Sci.* 34, 2/3 (1987), 266–292. https://doi.org/10.1016/0022-0000(87)90027-4

Ralph Matthes and Tarmo Uustalu. 2004. Substitution in non-wellfounded syntax with variable binding. *Theor. Comput. Sci.* 327, 1-2 (2004), 155–174. https://doi.org/10.1016/j.tcs.2004.07.025

James McKinna and Robert Pollack. 1999. Some Lambda Calculus and Type Theory Formalized. *J. Autom. Reason.* 23, 3-4 (1999), 373–409. https://doi.org/10.1023/A:1006294005493

Lawrence S. Moss. 2001. Parametric corecursion. *Theor. Comput. Sci.* 260, 1-2 (2001), 139–163. https://doi.org/10.1016/S0304-3975(00)00126-2

Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science, Vol. 2283. Springer. https://doi.org/10.1007/3-540-45949-9

Michael Norrish. 2004. Recursive Function Definition for Types with Binders. In *Theorem Proving in Higher Order Logics (TPHOLs) 2004*, Konrad Slind, Annette Bunker, and Ganesh Gopalakrishnan (Eds.). LNCS, Vol. 3223. Springer, 241–256. https://doi.org/10.1007/978-3-540-30142-4_18

Michael Norrish and René Vestergaard. 2007. Proof Pearl: De Bruijn Terms Really Do Work. In *Theorem Proving in Higher Order Logics, 20th International Conference, TPHOLs 2007, Kaiserslautern, Germany, September 10-13, 2007, Proceedings (Lecture Notes in Computer Science)*, Klaus Schneider and Jens Brandt (Eds.), Vol. 4732. Springer, 207–222. https://doi.org/10.1007/978-3-540-74591-4_16

Frank Pfenning and Carsten Schürmann. 1999. System Description: Twelf—A Meta-Logical Framework for Deductive Systems. In *Conference on Automated Deduction (CADE) 1999*, Harald Ganzinger (Ed.). LNCS, Vol. 1632. Springer, 202–206. https://doi.org/10.1007/3-540-48660-7_14

Brigitte Pientka. 2010. Beluga: Programming with Dependent Types, Contextual Data, and Contexts. In *Functional and Logic Programming (FLOPS) 2010*, Matthias Blume, Naoki Kobayashi, and Germán Vidal (Eds.). LNCS, Vol. 6009. Springer, 1–12. https://doi.org/10.1007/978-3-642-12251-4_1

Andrew M. Pitts. 2006. Alpha-Structural Recursion and Induction. *J. ACM* 53, 3 (2006), 459–506. https://doi.org/10.1145/1147954.1147961

Andrew M. Pitts. 2013. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press. https://doi.org/10.1017/CBO9781139084673

Andrew M. Pitts. 2023. Locally Nameless Sets. *Proc. ACM Program. Lang.* 7, POPL (2023), 488–514. https://doi.org/10.1145/3571210

Randy Pollack, Masahiko Sato, and Wilmer Ricciotti. 2012. A Canonical Locally Named Representation of Binding. *J. Autom. Reason.* 49, 2 (2012), 185–207. https://doi.org/10.1007/S10817-011-9229-Y

Andrei Popescu. 2023a. Nominal Recursors as Epi-Recurors (Mechanized Proofs Artifact). Zenodo, 2023. https://doi.org/10.5281/zenodo.10116628

Andrei Popescu. 2023b. Nominal Recursors as Epi-Recursors: Extended Technical Report. arXiv:2301.00894 [cs.LO] https://arxiv.org/abs/2301.00894.

Andrei Popescu. 2023c. Rensets and Renaming-Based Recursion for Syntax with Bindings: Extended Version. *J. Autom. Reason.* 67, 3 (2023), 23. https://doi.org/10.1007/S10817-023-09672-4

Andrei Popescu and Elsa L. Gunter. 2011. Recursion principles for syntax with bindings and substitution. In *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011*, Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy (Eds.). ACM, 346–358. https://doi.org/10.1145/2034773.2034819

Carsten Schürmann, Joëlle Despeyroux, and Frank Pfenning. 2001. Primitive recursion for higher-order abstract syntax. *Theor. Comput. Sci.* 266, 1-2 (2001), 1–57. https://doi.org/10.1016/S0304-3975(00)00418-7

Sam Staton. 2007. *Name-Passing Process Calculi: Operational Models and Structural Operational Semantics*. Technical Report UCAM-CL-TR-688. University of Cambridge, Computer Laboratory. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-688.pdf

Val Tannen and Ramesh Subrahmanyam. 1991. Logical and Computational Aspects of Programming with Sets/Bags/Lists. In *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings (Lecture Notes in Computer Science)*, Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo (Eds.), Vol. 510. Springer, 60–75. https://doi.org/10.1007/3-540-54233-7_125

Christian Urban and Stefan Berghofer. 2006. A Recursion Combinator for Nominal Datatypes Implemented in Isabelle/HOL. In *International Joint Conference on Automated Reasoning (IJCAR) 2006*, Ulrich Furbach and Natarajan Shankar (Eds.). LNCS, Vol. 4130. Springer, 498–512. https://doi.org/10.1007/11814771_41

Christian Urban, Stefan Berghofer, and Michael Norrish. 2007. Barendregt's Variable Convention in Rule Inductions. In *Conference on Automated Deduction (CADE) 2007*, Frank Pfenning (Ed.). LNCS, Vol. 4603. Springer, 35–50. https://doi.org/10.1007/978-3-540-73595-3_4

Christian Urban and Cezary Kaliszyk. 2012. General Bindings and Alpha-Equivalence in Nominal Isabelle. *Logical Methods in Computer Science* 8, 2 (2012). https://doi.org/10.2168/LMCS-8(2:14)2012

Christian Urban and Christine Tasson. 2005. Nominal Techniques in Isabelle/HOL. In *Conference on Automated Deduction (CADE) 2005*, Robert Nieuwenhuis (Ed.). LNCS, Vol. 3632. Springer, 38–53. https://doi.org/10.1007/11532231_4

Received 2023-07-11; accepted 2023-11-07