Barendregt Convenes with Knaster and Tarski: Strong Rule Induction for Syntax with Bindings

JAN VAN BRÜGGE, Heriot-Watt University, United Kingdom
JAMES MCKINNA, Heriot-Watt University, United Kingdom
ANDREI POPESCU, University of Sheffield, United Kingdom
DMITRIY TRAYTEL, University of Copenhagen, Denmark

This paper is a contribution to the meta-theory of systems featuring syntax with bindings, such as λ-calculi and logics. It provides a general criterion that targets *inductively defined rule-based systems*, enabling for them inductive proofs that leverage *Barendregt's variable convention* of keeping the bound and free variables disjoint. It improves on the state of the art by (1) achieving high generality in the style of Knaster–Tarski fixed point definitions (as opposed to imposing syntactic formats), (2) capturing systems of interest without modifications, and (3) accommodating infinitary syntax and non-equivariant predicates.

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

Additional Key Words and Phrases: syntax with bindings, induction, formal reasoning, nominal sets

ACM Reference Format:
Jan van Brügge, James McKinna, Andrei Popescu, and Dmitriy Traytel. 2025. Barendregt Convenes with Knaster and Tarski: Strong Rule Induction for Syntax with Bindings . *Proc. ACM Program. Lang.* 0, POPL, Article 0 (2025), 74 pages. https://doi.org/0

1 INTRODUCTION

Inductive definitions and proofs are a cornerstone of mathematics and theoretical computer science, and therefore solid and flexible foundations for induction are crucial in the development of these subjects—especially when it comes to the rigorous formulations and proofs of the results, using tools such as proof assistants. This paper is concerned with the formal foundations of induction for rule-based systems. Consider the basic example predicate describing whether a natural number is even:

$$\mathit{even}\ 0\ \text{(Base)} \qquad\qquad \frac{\mathit{even}\ n}{\mathit{even}\ (n+2)}\ \text{(Ind)}$$

The definition is inductive: a base case states that 0 is even, and an inductive case states that $n + 2$ is even if n is even. The intention is that all even numbers, and only those, are obtained by repeated application of the two rules; or equivalently, *even* is the smallest predicate closed under these rules.

One can take a *syntactic-format* approach to making sense of this and similar definitions, by proving a theorem such as: "For any specification consisting of rules where the conclusion and the hypotheses say that the to-be-defined inductive predicate applied to some arguments holds true, there exists the smallest predicate that satisfies the specification." Various relaxations and enhancements

Authors' addresses: Jan van Brügge, jsv2000@hw.ac.uk, Department of Computer Science, Heriot-Watt University, Edinburgh, United Kingdom; James McKinna, j.mckinna@hw.ac.uk, Department of Computer Science, Heriot-Watt University, Edinburgh, United Kingdom; Andrei Popescu, School of Computer Science, University of Sheffield, Sheffield, United Kingdom, a.popescu@sheffield.ac.uk; Dmitriy Traytel, Department of Computer Science, University of Copenhagen, Copenhagen, Denmark, traytel@di.ku.dk.

ACM 2475-1421/2025/0-ART0
https://doi.org/0

of such a format are possible, e.g., allowing non-recursive assumptions, side-conditions, and specifying a grammar for the arguments to which the to-be-defined predicate is applied. But no matter how far we go with format enhancements, we are likely to encounter situations where they are still not enough. Particularly difficult aspects to capture via formats are nested quantifiers and higher-order operators. For example, consider the set *Tree* of finite trees whose leaves *Leaf* are labelled by natural numbers and such that every tree $t \in \mathit{Tree}$ has a finite (possible empty) set $\mathit{Desc}\ t \in \mathcal{P}_{\mathrm{fin}}(\mathit{Tree})$ of immediate descendants. We can define inductively the following parity simulation relation $\preceq$ on trees:

$$\frac{t = \mathit{Leaf}\ n \qquad t' = \mathit{Leaf}\ (2 * n)}{t \preceq t'}\ (\mathrm{Base}) \qquad \frac{\mathit{isDesc}\ t \qquad \mathit{isDesc}\ t' \qquad \mathit{RelSet}\ (\preceq)\ (\mathit{Desc}\ t)\ (\mathit{Desc}\ t')}{t \preceq t'}\ (\mathrm{Ind})$$

where $\mathit{isDesc}\ t$ states that t is not a leaf and, for any binary relation R on a set A, $\mathit{RelSet}\ R$ denotes its Hoare-style extension to a relation on $\mathcal{P}_{\mathrm{fin}}(A)$ defined by $\mathit{RelSet}\ R\ B\ B' = (\forall a \in B.\ \exists a' \in B'.\ R\ a\ a')$.

For making sense of rule-based inductive definitions, an approach that is more general and principled (and conceptually simpler!) than the syntactic-format approach is possible, by noticing that the existence of a smallest predicate satisfying a specification is guaranteed *regardless of its format*, provided it can be expressed using a monotonic operator on predicates. The operators underlying the definitions of *even* and $\preceq$ are G_{even} and $G_{\preceq}$ are defined as follows:

$G_{even}\ P\ m = (m = 0\ \vee\ \exists n.\ m = n + 2 \wedge P\ n)$

$G_{\preceq}\ R\ t\ t' = ((\exists n.\ t = \mathit{Leaf}\ n \wedge t' = \mathit{Leaf}\ (2{*}n)) \vee (\mathit{isDesc}\ t \wedge \mathit{isDesc}\ t' \wedge \mathit{RelSet}\ R\ (\mathit{Desc}\ t)\ (\mathit{Desc}\ t')))$

For any monotonic operator on a complete lattice (such as the lattice of predicates), as is easily seen to be the case with G_{even} and $G_{\preceq}$, the Knaster–Tarski theorem [Tarski 1955] guarantees the existence of a least fixed point. So *even* and $\preceq$ both exist as the least fixed points of G_{even} and $G_{\preceq}$, and have the desired properties, including induction principles for reasoning about them, merely by virtue of these operators being monotonic. The precise format of the predicate does not matter. In particular, for $\preceq$ the definition of *RelSet* is irrelevant, other than it is monotonic. This monotonicity-based approach was a major breakthrough, since it covers both existing and future syntactic formats that one would be interested in. It was implemented as part of the induction facilities of several proof assistants, notably the ones based on higher-order logic including HOL4 [Gordon and Melham 1993], HOL Light [Harrison 2024] and Isabelle/HOL [Nipkow et al. 2002].

Here we will be concerned with inductive definitions involving syntax with bindings—pervasive in the theory of logics and programming languages, where variables are being bound in terms and formulas via quantifiers, λ-abstractions, etc. When working with these systems, researchers want to avoid the overlap between bound and free variables, lest their proofs become significantly harder or fail altogether. This means applying 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."

This informal principle has been made rigorous by subsequent research, notably in the context of Nominal Logic [Gabbay and Pitts 1999, 2002] and related formalisms (e.g., [Aydemir et al. 2008]). Specifically for inductive rule-based systems involving binders, Urban et al. [2007] identified a rule format and some assumptions that are sufficient for allowing Barendregt's variable convention to be soundly used in proofs, leading to a *strong induction principle* criterion guaranteeing the disjointness between bound and free variables. Subsequently, this criterion has been implemented as part of the Nominal Isabelle package [Urban 2008; Urban and Kaliszyk 2012].

A natural question to ask is whether **(1)** a more general, monotonicity-based approach (that does not require a syntactic format) can be pursued here. Besides the limitations stemming from the syntactic format, another limitation of the state of the art is that **(2)** it fails to directly capture existing mainstream systems such as the λ-calculus reduction and π-calculus transition relations, but requires the modification of these systems' standard presentations by adding more side-conditions. In other words, there is a gap between the standard definitions of these systems from textbooks and the formal requirements for enabling the variable convention. Finally, the state of the art, deeply rooted in Nominal Logic and its finite support and equivariance conditions (the latter expressing a form of uniform behavior of functions and predicates) [Gabbay and Pitts 2002; Pitts 2006], **(3)** does not cover infinitary syntax with bindings such as infinitary extensions of the λ-calculus [Barendregt and Klop 2009; Mazza 2012] and first-order logic [Hanf 1964; Makkai 1969].

This paper makes contributions along all the above three axes. It introduces general criteria for when inductive systems are variable-convention observing, leveraging monotonicity and Knaster–Tarski. Our criteria also fill the aforementioned formality gap as they apply to the systems without modifications, and moreover cope with infinitary syntax and the lack of equivariance.

Overview. We start with revisiting standard examples coming from the λ-calculus (§2), highlighting the limitations of the state of the art. Then, after recalling the necessary background concepts pertaining to nominal sets and induction (§3), we prove the initial version of our main result, a format-free general criterion for strong rule induction (§4). This initial version will be further improved and generalized throughout the rest of the paper by challenging it with inductive systems whose syntactic structures or binding dynamics are increasingly sophisticated. We first deploy our criterion to tackle the motivating examples (§5), which leads us to a deployment heuristic (§6). We compare our criterion with the state of the art criterion of Urban et al. [2007] with respect to the addition of side-conditions (§7). More examples are discussed (§8), including the π-calculus and subtyping for System $F_{<:}$, the latter suggesting a strengthening of our criterion with inductive information. Further examples take us into the realm of infinitary structures with bindings (§9), such as extensions of first-order logic that allow infinitary cardinal-bounded conjunctions and quantifications in formulas (§9.1). To extend our criterion for coping with predicates defined over infinitary structures, we introduce what we call *loosely-supported nominal sets* (§9.2), a variation of nominal sets equipped with a "loose" (not necessarily minimal) supporting set operator that relax the finite-support assumption to a small-support one, where "small" is understood with respect to a given infinite cardinal. The last example we consider involves the meta-theory of an affine infinitary λ-calculus (§9.3), and leads to a further generalization of our criterion to handle non-equivariant predicates (§9.4). We describe a tool that we have implemented in Isabelle to support our formalization of the general theory and the examples, as well as case studies based on these examples (§10), and conclude with more related work (§11). An appendix gives more details about this paper's constructions and results, and our Isabelle mechanization.

2 MOTIVATING EXAMPLE: λ-CALCULUS

In this section, *Var*, the set of variables, will be a countably infinite set. We consider the syntax of the (untyped) λ-calculus, defining the set *LTerm* of *λ-terms*, ranged over by t, s etc., via the grammar:

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

Thus, a λ-term is either (the injection of) a variable, or an application, or a λ-abstraction of a variable in a term. We also assume that, in a term of the form $Lm\ x\ t$, the variable x is bound in t; and terms are equated modulo the induced notion of alpha-equivalence, e.g., $Lm\ x\ (Vr\ x) = Lm\ y\ (Vr\ y)$. For any λ-term t, we write $FV\ t$ for its set of *free variables*. A variable x is *fresh* for t when $x \notin FV\ t$. We write $t[s/x]$ for the (capture-avoiding) *substitution* of the term s for the variable x in the term t.

A fundamental relation on this syntax is *β-reduction*, the binary relation $\Rightarrow$ between λ-terms defined in Fig. 1. If we ignore binding information, the standard proof principle associated to this definition is the following *rule induction* principle:

Prop 1. Let $\varphi : LTerm \to LTerm \to Bool$ and assume that:

- (|Beta|): $\forall x, t_1, t_2.\ \varphi\ (Ap\ (Lm\ x\ t_1)\ t_2)\ (t_1[t_2/x])$

$$Ap\ (Lm\ x\ t_1)\ t_2 \Rightarrow t_1[t_2/x]\ \text{(Beta)} \qquad \frac{t \Rightarrow t'}{Lm\ x\ t \Rightarrow Lm\ x\ t'}\ \text{(Xi)}$$

$$\frac{t_1 \Rightarrow t_1'}{Ap\ t_1\ t_2 \Rightarrow Ap\ t_1'\ t_2}\ \text{(ApL)} \qquad \frac{t_2 \Rightarrow t_2'}{Ap\ t_1\ t_2 \Rightarrow Ap\ t_1\ t_2'}\ \text{(ApR)}$$

Fig. 1. λ-calculus β-reduction

- (|Xi|): $\forall x, t, t'.\ ((t \Rightarrow t') \wedge \varphi\ t\ t') \longrightarrow \varphi\ (Lm\ x\ t)\ (Lm\ x\ t')$
- (|ApL|): $\forall t_1, t_1', t_2.\ \big((t_1 \Rightarrow t_1') \wedge \varphi\ t_1\ t_1'\big) \longrightarrow \varphi\ (Ap\ t_1\ t_2)\ (Ap\ t_1'\ t_2)$
- (|ApR|): $\forall t_1, t_2, t_2'.\ \big((t_2 \Rightarrow t_2') \wedge \varphi\ t_2\ t_2'\big) \longrightarrow \varphi\ (Ap\ t_1\ t_2)\ (Ap\ t_1\ t_2')$

Then $\forall t, t'.\ (t \Rightarrow t') \longrightarrow \varphi\ t\ t'$.

Thus, standard induction allows us to infer that $\Rightarrow$ is included in a relation φ provided φ is closed under the rules defining $\Rightarrow$, i.e., uses that $\Rightarrow$ is the smallest relation closed under these rules.

However, due to the presence of bindings, it is desirable to have a stronger induction proof principle—featuring an enhancement that formalizes Barendregt's variable convention. For example, say we want to prove that β-reduction is closed under substitution, i.e, $(t \Rightarrow t') \longrightarrow (t[s/y] \Rightarrow t'[s/y])$. The proof would go by rule induction, taking $\varphi\ t\ t'$ to be $\forall s, y.\ t[s/y] \Rightarrow t'[s/y]$. In the (|Beta|) case we must prove $\varphi\ (Ap\ (Lm\ x\ t_1)\ t_2)\ (t_1[t_2/x])$, i.e., for all s, y,

$$\textbf{(i)}\ \ (Ap\ (Lm\ x\ t_1)\ t_2)[s/y] \Rightarrow t_1[t_2/x][s/y]$$

To continue, we wish to move the $\_[s/y]$ substitution inside the constructors Ap and Lm on the left, and also inside the $t_1[t_2/x]$ substitution on the right, thus reducing the above to

$$\textbf{(ii)}\ \ Ap\,(Lm\,x\,(t_1[s/y]))\,(t_2[s/y]) \Rightarrow (t_1[s/y])[(t_2[s/y])/x]$$

the last being provable as an instance of the (Beta) rule, taking t_1 and t_2 from the rule to be $t_1[s/y]$ and $t_2[s/y]$. (Without being able to perform the above "moves", the proof would become quite complicated, as the goal would need to be generalized to work inductively.)

However, while substitution can soundly be moved inside applications (since by definition it commutes with applications), it is not always sound to move it inside λ-abstractions or other substitutions, *unless certain side-conditions hold.* In this case, we would need that x *is fresh for the parameters*, i.e., x is fresh for s and is different from y, which would ensure $(Lm\ x\ t_1)[s/y] = Lm\ x\ (t_1[s/y])$ and $t_1[t_2/x][s/y] = (t_1[s/y])[(t_2[s/y])/x]$, making (i) reducible to (ii) as desired for finishing the proof in the (|Beta|) case. Barendregt's insight, expressed in his variable convention and deployed systematically in proofs all throughout his λ-calculus monograph [Barendregt 1985], was that such freshness assumptions are usually safe, in that they do not lose generality (hence do not lead to incorrect reasoning).

Yet, Barendregt did not indicate exactly *when*, or *why*, such assumptions are safe. A rigorous answer to these questions was provided by Urban et al. [2007] (having prior roots in McKinna and Pollack [1999]; Pitts [2003]) who formalized the variable convention used in proof contexts like the above as a *strong rule induction* that allows assuming the rules' bound variables (e.g., x in (Beta) and (Xi)) to be fresh for given parameters (e.g., s and y). Here is the desired strong rule induction for β-reduction, where $\mathcal{P}_{\mathsf{fin}}(Var)$ is the set of finite sets of variables:

Prop 2. Let P be a set of items called *parameters* and $Psupp : P \to \mathcal{P}_{\mathsf{fin}}(Var)$. Let $\varphi : P \to LTerm \to LTerm \to Bool$ and assume that:

- (|Beta|): $\forall p, x, t_1, t_2.\ x \notin Psupp\ p \longrightarrow \varphi\ p\ (Ap\ (Lm\ x\ t_1)\ t_2)\ (t_1[t_2/x])$
- (|Xi|): $\forall p, x, t, t'.\ x \notin Psupp\ p \wedge (t \Rightarrow t') \wedge (\forall q.\ \varphi\ q\ t\ t') \longrightarrow \varphi\ p\ (Lm\ x\ t)\ (Lm\ x\ t')$
- (|ApL|): $\forall p, t_1, t_1', t_2.\ (t_1 \Rightarrow t_1') \wedge (\forall q.\ \varphi\ q\ t_1\ t_1') \longrightarrow \varphi\ p\ (Ap\ t_1\ t_2)\ (Ap\ t_1'\ t_2)$
- (|ApR|): $\forall p, t_1, t_2, t_2'.\ (t_2 \Rightarrow t_2') \wedge (\forall q.\ \varphi\ q\ t_2\ t_2') \longrightarrow \varphi\ p\ (Ap\ t_1\ t_2)\ (Ap\ t_1\ t_2')$

Then $\forall p, t, t'.\ (t \Rightarrow t') \longrightarrow \varphi\ p\ t\ t'$.

The predicate to be proved is now quantified universally over parameters, whose role is to provide the variables that one would like to avoid within inductive proofs—what Barendregt's convention, cited in the introduction, calls "the free variables" in a "certain mathematical context". To use this principle in the proof discussed above, we take P to be $LTerm \times Var$ and $Psupp(s, y)$ to be $FV\ s \cup \{y\}$. (Note that a weaker form of this principle would fix a parameters p rather than quantifying universally over parameters, so that φ would not have a parameter argument and, for example, the hypothesis (|Xi|) would become $\forall x, t, t'.\ x \notin Psupp\ p \wedge (t \Rightarrow t') \wedge \varphi\ t\ t' \longrightarrow \varphi\ (Lm\ x\ t)\ (Lm\ x\ t')$ and (|ApL|) and (|ApL|) would become the usual inductive conditions from the standard rule induction expressed by Prop. 1. While often the fixed-parameter version is good enough, as is the case with the proof discussed above which works for fixed s and y, sometimes the extra flexibility of quantifying universally over parameters is important—Lemma 107 from App. F (reflexivity of the System $F_{<:}$ typing from POPLmark 1A [Aydemir et al. 2005]) gives an example for structural induction which is a particular case of rule induction.)

Importantly, Urban et al. [2007] have also noted that Barendregt's variable convention is not sound for all inductively defined relations on λ-terms, and have provided a syntactic criterion for when it *is* sound. Unfortunately, their criterion does not cover the above (standard) definition of β-reduction (shown in Fig. 1) but only a modification of it obtained by adding a freshness side-condition to the (Beta) rule:

$$Ap\ (Lm\ x\ t_1)\ t_2 \Rightarrow t_1[t_2/x]\ \begin{array}{l}\text{(Beta')}\\ [x \notin FV\ t_2]\end{array}$$

With this modification, strong induction for β-reduction, i.e., Prop. 2, becomes an instance of their syntactic criterion. This variant of β-reduction, with (Beta') instead of (Beta), can be proved equivalent to the standard one, but this is far from immediate.

The need for adding side-conditions arises quite pervasively when instantiating Urban et al.'s result to examples. In fact, the authors themselves show such an example in their paper: a parallel β-reduction [Lévy 1975; Takahashi 1995], where they must change the "Parallel Beta" rule

$$\frac{t_1 \Longrightarrow t_1' \qquad t_2 \Longrightarrow t_2'}{Ap\ (Lm\ x\ t_1)\ t_2 \Longrightarrow t_1'[t_2'/x]}\ \text{(ParBeta)}$$

into the weaker rule

$$\frac{t_1 \Longrightarrow t_1' \qquad t_2 \Longrightarrow t_2'}{Ap\ (Lm\ x\ t_1)\ t_2 \Longrightarrow t_1'[t_2'/x]}\ \begin{array}{l}\text{(ParBeta')}\\ [x \notin FV\ t_2 \cup FV\ t_2']\end{array}$$

Quoting from Urban et al.: "This is annoying because both versions can be shown to define the same relation, but we have no general, and automatable, method for determining this."

Another limitation of their criterion (again acknowledged by the authors themselves) is its syntactic-format nature, requiring rules the form

$$\frac{\varphi\ p_1\ \vec{s_1} \qquad \ldots \qquad \varphi\ p_n\ \vec{s_n}}{\varphi\ p\ \vec{t}}\,[\text{side-conditions}]$$

which is quite rigid. In particular this forbids, in the rules' assumptions, the occurrence of the defined relation under universal or existential quantifiers, or under other higher-order operators. Our results will lift both of the above limitations.

3 PRELIMINARIES ON NOMINAL SETS AND KNASTER–TARSKI FIXPOINTS

Next we recall some background on nominal sets [Gabbay and Pitts 2002; Pitts 2013] and induction based on Knaster–Tarski fixpoints [Knaster 1928; Tarski 1955].

Nominal sets. Let *Var* be a fixed countable sets of items called *variables*, or *atoms*. Given any function $f : \mathit{Var} \to \mathit{Var}$, the *core* of f is defined as the set of all variables that are changed by f: $\mathit{Core}\ f = \{x \mid f\ x \neq x\}$. (What we call "core" is usually called the "support" of f, which is consistent with the more general notion of support we discuss next. But we prefer to name it differently because of its bootstrapping role towards the general notion.) Let *Perm*, ranged over by σ, denote the set of (finite) *permutations*, i.e., bijections on *Var* of finite core.

Note that $(\mathit{Perm}, \circ, 1_{\mathit{Var}})$ forms a group, where 1_{Var} 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 $\_[\_]^{\mathcal{A}} : A \to \mathit{Perm} \to A$ is an action of the monoid *Perm* on A, in that it is idle for identity ($a[1_{\mathit{Var}}]^{\mathcal{A}} = a$ for all $a \in A$) and compositional ($a[\sigma \circ \tau]^{\mathcal{A}} = a[\tau]^{\mathcal{A}}[\sigma]^{\mathcal{A}}$). Given $\sigma \in \mathit{Perm}$, we sometimes write $\_[\sigma]$ for the function in $A \to A$ (which is actually a bijection) that applies this fixed permutation. We let $\mathit{Im}\ \sigma$ be the operator in $\mathcal{P}(\mathit{Var}) \to \mathcal{P}(\mathit{Var})$ that takes any $X \subseteq \mathit{Var}$ to the image of X through σ, namely $\mathit{Im}\ \sigma\ X = \{\sigma\ x \mid x \in X\}$. We write $x \leftrightarrow y$ for the permutation that takes x to y, y to x and all other variables to themselves; applying these permutations (to elements of a nominal set) will be called *swapping*.

Given a pre-nominal set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$, an $a \in A$ and a set $X \subseteq \mathit{Var}$, we say that *a is supported by X*, or *X supports a*, if $a[x{\leftrightarrow}y]^{\mathcal{A}} = a$ holds for all $x, y \in \mathit{Var} \smallsetminus X$, or equivalently, if $a[\sigma]^{\mathcal{A}} = a$ holds for all $\sigma \in \mathit{Perm}$ such that $\forall x \in X.\ \sigma\ x = x$. 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 $\mathit{Supp}^{\mathcal{A}}\ a$ 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 naturally 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 $f(a[\sigma^{-1}])[\sigma]$. (So in particular we can talk about the notion of a set of variables supporting such a function.) $\mathcal{F}$ is not a nominal set, because not all functions are finitely supported, but we obtain a nominal set if we restrict it to the finitely supported functions. In addition to the above function-space construction, nominal set structures can also be naturally defined on the products, sums, container-type extensions (such as lists or trees) and quotients of the carrier sets, overall enjoying good category-theoretic properties, in particular forming a topos equivalent to the Schanuel topos [Pitts 2013].

The set of λ-terms with their standard *Perm*-action, $(\mathit{LTerm}, \_[\_])$, forms a nominal set, where the support of a term t consists of its free variables. Note that set $\mathit{FV}\ t$ of free variables of a λ-term t is traditionally defined recursively on the structure of t and not from permutation like the support is. However, writing *Supp* for the support operator of the nominal set $(\mathit{LTerm}, \_[\_])$, it can be checked that (1) t is supported by $\mathit{FV}\ t$ in that $t[x{\leftrightarrow}y] = t$ holds whenever $x, y \notin \mathit{FV}\ t$, by an easy induction on t; and that (2) for any x, assuming $x \in \mathit{FV}\ t \smallsetminus \mathit{Supp}\ t$ yields a contradiction by taking some $y \notin \mathit{FV}\ t \cup \mathit{Supp}\ t$ and noting that $t[x{\leftrightarrow}y] \neq t$ (which again follows by easy induction from $x \in \mathit{FV}\ t$ and $y \notin \mathit{FV}\ t$) contradicts the fact that $x, y \notin \mathit{Supp}\ t$. Points (1) and (2) make $\mathit{FV}\ t$ coincide with $\mathit{Supp}\ t$. It is known (and can be established by an argument similar to the one sketched above) that this coincidence between the free-variable operator and support holds for all syntaxes with statically scoped bindings [Pitts 2006], so any such syntax forms a nominal set where the support is given by the free variables.

Although the concept of nominal set abstracts away from, and goes beyond syntactic objects (covering for example restricted spaces of functions, of semantic entities etc. [Pitts 2013]), it is often useful to think of the elements a of a nominal set as "term-like" entities; in this spirit, we will refer to the elements of $\mathit{Supp}^{\mathcal{A}}\ a$ as the free-variables of a.

Nominal sets underpin the semantics of *nominal logic* [Gabbay and Pitts 1999, 2002], a successful foundation tailored for reasoning about syntax with bindings. But nominal sets and nominal logic techniques can also be used from within general-purpose foundations such as higher-order logic [Pitts 2006; Urban and Tasson 2005]—in this paper we subscribe to this approach.

Central in nominal logic, and in our own developments as well, is the notion of equivariance, which for a function, predicate or assertion means commutation with permutation actions. With roots in classical algebra [Pitts 2013, §1.1], equivariance has the following intuition in the context of syntax with bindings, as explained in the seminal nominal logic paper [Gabbay and Pitts 1999, §2]: "Properties of syntax should be sensitive only to distinctions between variable names, rather than to the particular names themselves." Here is the formal definition:

Def 3. Given two pre-nominal sets $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ and $\mathcal{B} = (B, \_[\_]^{\mathcal{B}})$, a function $F : A \to B$ between their carrier sets is called *equivariant* when it commutes with the permutation actions: $F\ (a[\sigma]^{\mathcal{A}}) = (F\ a)[\sigma]^{\mathcal{B}}$ for all $a \in A$ and $\sigma \in \mathit{Perm}$.

Since the two-element set of Booleans (like any set) can be trivially equipped with identity permutation action to become a (pre-)nominal set, we can speak of the equivariance of predicates, $\varphi : A \to \mathit{Bool}$ where $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ is a pre-nominal set. Here, equivariance can be equivalently expressed using implication: $\varphi\ a \longrightarrow \varphi\ (a[\sigma]^{\mathcal{A}})$ for all $a \in A$ and $\sigma \in \mathit{Perm}$.

The Knaster–Tarski Fixpoint Theorem. This celebrated result offers a simple yet powerful foundation for induction, with applications in areas such as semantics, verification and static analysis.

Thm 4. [Tarski 1955] Let $(L, \leq)$ be a complete lattice and $G : L \to L$ a monotonic operator. Then there exists a (unique) least fixpoint I_G for G, in that: $G\ I_G = I_G$ and $\forall k \in G.\ G\ k = k \longrightarrow I_G \leq k$. And I_G is the least pre-fixpoint as well, in that $\forall k \in L.\ G\ k \leq k \longrightarrow I_G \leq k$; finally, a practically useful variation of this also holds, where $\wedge$ is binary infimum in L: $\forall k \in L.\ G\ (I_G \wedge k) \leq k \longrightarrow I_G \leq k$.

It is the "pre-fixpoint" part of this theorem that enables inductive reasoning: To prove that $I_G \leq k$, it suffices to prove that $G\ (I_G \wedge k) \leq k$. While the theorem works in general for complete lattices, we will only use it for the particular lattices of predicates (equivalently, lattices of subsets), as initially formulated by Knaster [1928]. Given a set A and two predicates $\varphi, \psi : A \to \mathit{Bool}$ on it, we define $\varphi \leq \psi$ to be component-wise implication, namely $\forall a \in A.\ \varphi\ a \longrightarrow \psi\ a$. And indeed, $\leq$ is a complete-lattice order on the set $A \to \mathit{Bool}$ of predicates. This applies to n-ary predicates as well if we take A to be a product $A_1 \times \ldots \times A_n$. Often the operator G on predicates is given by a set of rules, and for this reason the emerging induction principle associated to I_G is referred to as *rule induction*.

4 STRONG RULE INDUCTION CRITERION

Our main result, which we present next (Thm. 7 below), is an extension of Knaster–Tarski based rule induction to strong (variable-convention observing) induction, leveraging nominal-set structure.

We start with a monotonic operator $G : (T \to \mathit{Bool}) \to (\mathcal{P}_{\mathsf{fin}}(\mathit{Var}) \to T \to \mathit{Bool})$, where monotonicity again refers to the standard predicate orderings (component-wise implication). We iterate G to define the predicate $I_G : T \to \mathit{Bool}$ inductively as follows: $\dfrac{G\ I_G\ B\ t}{I_G\ t}$

We think of the above as the inductive specification of a rule-based system I_G. But differently from the usual Knaster–Tarski setting for such specifications, here we have made explicit an additional "bound variable set" argument $B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var})$ for the predicate returned by G. Our strong rule induction criterion will make assumptions on, and draw conclusions from, how G operates B.

But first let us make sense of the above specification of I_G without treating B specially. That I_G was obtained by "iterating" G means that I_G is the least (pre-)fixpoint of the operator $\lambda\,\varphi.\ \lambda\,t \in T.\ \exists B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var}).\ G\ \varphi\ B\ t$. Its existence is guaranteed by Thm. 4, taking I_G to be the least fixpoint of the operator on $(T \to \mathit{Bool}) \to (T \to \mathit{Bool})$ that acts like G but applies existential quantification over B, i.e., sends any predicate $\varphi : T \to \mathit{Bool}$ to $\lambda t.\ \exists B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var}).\ G\ \varphi\ B\ t$. The standard rule induction principle stemming from I_G's definition (via Thm. 4) is the following:

Thm 5. Assume G is monotonic. If $\varphi : T \to \mathit{Bool}$ is such that $\forall t \in T.\ (\exists B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var}).\ G\ (\lambda t'. I_G\, t' \wedge \varphi\ t')\ B\ t) \longrightarrow \varphi\ t$, then $I_G \leq \varphi$, i.e., $\forall t \in T.\ I_G\, t \longrightarrow \varphi\ t$.

(The assumption of Thm. 5 is equivalent to $\forall t \in T, \forall B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var}).\ G\ (\lambda t'. I_G\, t' \wedge \varphi\ t')\ B\ t \longrightarrow \varphi\ t$.)

Now let us make our move towards strong rule induction. To formulate such a principle without knowing how G looks like, we think of G as the rules defining our predicate I_G; and of its argument B as the bound variables appearing in the *conclusions* of these rules—for this interpretation to make sense, we assume that I_G operates on "term-like" entities, i.e., elements of a nominal set $\mathcal{T}$. Our key observation is that Barendregt's variable convention rests on the bound variables being "refreshable" in the rules, in that (roughly speaking) we can always rename them to become fresh for a rule's *entire* conclusion (and not just the location where they are bound) *without invalidating its hypotheses*. To model this, we introduce the concept of $\mathcal{T}$-refreshability; and also introduce $\mathcal{T}$-freshness, which goes further to say that the bound variables are already fresh.

Def 6. Given a nominal set $\mathcal{T} = (T, \_[\_]^{\mathcal{T}})$, an operator $G : (T \to \mathit{Bool}) \to (\mathcal{P}_{\mathsf{fin}}(\mathit{Var}) \to T \to \mathit{Bool})$ is said to be:
- *$\mathcal{T}$-refreshable* when, for all $\varphi : T \to \mathit{Bool}$, $B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var})$ and $t \in T$, if φ is equivariant and $G\ \varphi\ B\ t$ then there exists $B' \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var})$ such that $B' \cap \mathit{Supp}^{\mathcal{T}} t = \emptyset$ and $G\ \varphi\ B'\ t$;
- *$\mathcal{T}$-fresh* when, for all $\varphi : T \to \mathit{Bool}$, $B \in \mathcal{P}_{\mathsf{fin}}\ (\mathit{Var})$ and $t \in T$, if $G\ \varphi\ B\ t$ then $B \cap \mathit{Supp}^{\mathcal{T}} t = \emptyset$.

(Note that $\mathcal{T}$-freshness implies $\mathcal{T}$-refreshability, taking $B' = B$.)

And indeed, we can prove that $\mathcal{T}$-refreshability in conjunction with equivariance (which essentially ensures robustness of the rules in the refreshing process) is sufficient for enabling strong rule induction. In what follows, a pair (P, Psupp) where P is a set and $\mathit{Psupp} : P \to \mathcal{P}_{\mathsf{fin}}(\mathit{Var})$ will be called *parameter structure*. (These are not required to be nominal sets.)

Thm 7. Let $\mathcal{T} = (T, \_[\_]^{\mathcal{T}})$ be a nominal set and $G : (T \to \mathit{Bool}) \to (\mathcal{P}_{\mathsf{fin}}(\mathit{Var}) \to T \to \mathit{Bool})$ a monotonic, equivariant and $\mathcal{T}$-refreshable operator. Let (P, Psupp) be a parameter structure and $\varphi : P \to T \to \mathit{Bool}$ a predicate. Assume that:

$$\forall p \in P, t \in T, B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var}).\ \left(\begin{array}{c} B \cap (\mathit{Psupp}\ p \cup \mathit{Supp}^{\mathcal{T}} t) = \emptyset\ \wedge \\ G\ (\lambda t'.\ I_G\, t' \wedge \forall p' \in P.\ \varphi\ p'\ t')\ B\ t \end{array} \right) \longrightarrow \varphi\ p\ t$$

Then $\forall p \in P.\ I_G \leq \varphi\ p$, i.e., $\forall p \in P, t \in T.\ I_G\, t \longrightarrow \varphi\ p\ t$.

Highlighted above is the "strength" of the stated strong induction principle for I_G: When performing induction, we are allowed to assume the variables of the parameter p, and also the free variables of the nominal-set (i.e., the term-like entity) argument t, to be distinct from the variables in B (the bound variables). In short, the bound variables can be avoided.

We show a detailed proof of this result, partly because we will later do a bit of proof mining for generalizing it. The main idea is that, using $\mathcal{T}$-refreshability, we are able to "clean up" the inductive definition of I_G to assume freshness of the bound-variables B for the rules' conclusions t (i.e., $B \cap \mathit{Supp}^{\mathcal{T}} t = \emptyset$), and then use G's equivariance to prove that freshness for the parameters can also be assumed.

Proof. We will write $\_[\_]$ instead of $\_[\_]^{\mathcal{T}}$ and Supp instead of $\mathit{Supp}^{\mathcal{T}}$.

We first define an inductive predicate I'_G which is a variation of I_G that factors in "half" of the intended freshness assumption, namely $B \cap \mathit{Supp}\ t = \emptyset$:

$$\frac{G\ I'_G\ B\ t \qquad B \cap \mathit{Supp}\ t = \emptyset}{I'_G\ t}$$

Since the defining rule for I'_G is weaker (has more hypotheses), I'_G is stronger than I_G, we have: **(1)** $\forall t.\ I'_G\ t \longrightarrow I_G\ t$. Crucially, we will be able to also prove the converse of (1). But first we need: **(2)** I'_G is equivariant, i.e., $\forall \sigma \in \mathit{Perm},\ t \in T.\ I'_G\ t \longrightarrow I'_G\ (t[\sigma])$.

The proof of (2) goes by rule induction on the definition of I'_G, for an (arbitrary but) fixed $\sigma \in \mathit{Perm}$: We fix $B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var})$ and $t \in T$ and assume **(i)** $G\ (I'_G \circ (\_[\sigma]))\ B\ t$ and **(ii)** $B \cap \mathit{Supp}\ t = \emptyset$. We must show that $I'_G\ (t[\sigma])$. Using the introduction rule associated to the definition of I'_G, it suffices to show **(i')** $G\ I'_G\ (\mathit{Im}\ \sigma\ B)\ (t[\sigma])$ and **(ii')** $\mathit{Im}\ \sigma\ B \cap \mathit{Supp}\ (t[\sigma]) = \emptyset$. From (i) and the equivariance of G, we obtain $G\ (I'_G \circ (\_[\sigma]) \circ (\_[\sigma^{-1}])\ (\mathit{Im}\ \sigma\ B)\ (t[\sigma])$, hence, by the fact that $\sigma \circ \sigma^{-1} = 1_{\mathit{Var}}$ and the functoriality of $\_[\_]$, we obtain (i'), as desired. Moreover, (ii') follows from (ii) and the properties of *Supp*. (Note that so far we used G's equivariance and monotonicity, but not yet its $\mathcal{T}$-refreshability.)

Now we prove **(3)** $\forall t.\ I_G\ t \longrightarrow I'_G\ t$, by rule induction on the definition of I_G: We fix $B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var})$ and $t \in T$ and assume (iii) $G\ I'_G\ B\ t$. We must show $I'_G\ t$. From (2), (iii) and $\mathcal{T}$-refreshability, we obtain $B' \in \mathcal{P}_{\mathsf{fin}}\ (\mathit{Var})$ such that $B' \cap \mathit{Supp}\ t = \emptyset$ and $G\ I'_G\ B'\ t$. Hence, $I'_G\ t$ follows by I'_G's introduction rule.

From (1) and (3), we have **(4)** $I_G = I'_G$. Now we are ready to tackle the theorem's statement, in which, using (4), we will freely replace I_G with I'_G. Thus, we assume

$$\textbf{(5)}\ \forall p \in P, t \in T, B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var}).\ \begin{pmatrix} B \cap (\mathit{Psupp}\ p \cup \mathit{Supp}\ t) = \emptyset\ \wedge \\ G\ (\lambda t'.\ I'_G\ t' \wedge \forall p' \in P.\ \varphi\ p'\ t')\ B\ t \end{pmatrix} \longrightarrow \varphi\ p\ t$$

We must prove $\forall p \in P,\ t \in T.\ I'_G\ t \longrightarrow \varphi\ p\ t$, i.e., $\forall t \in T.\ I'_G\ t \longrightarrow (\forall p \in P.\ \varphi\ p\ t)$. We will prove something more general, namely that I'_G implies the equivariant envelope of φ: $\forall t \in T.\ I'_G\ t \longrightarrow (\forall \sigma \in \mathit{Perm}.\ \forall p \in P.\ \varphi\ p\ (t[\sigma]))$.

We again proceed by rule induction on the definition of I'_G: We fix $B \in \mathcal{P}_{\mathsf{fin}}(\mathit{Var})$, $t \in T$, $\sigma \in \mathit{Perm}$ and $p \in P$ and assume **(iv)** $G\ (\lambda t'.\ I'_G\ t' \wedge (\forall \sigma' \in \mathit{Perm},\ p' \in P.\ \varphi\ p'\ (t'[\sigma'])))\ B\ t$ and **(v)** $B \cap \mathit{Supp}\ t = \emptyset$. We must show $\varphi\ p\ (t[\sigma])$.

Let $B' = \mathit{Im}\ \sigma\ B$. Note that B' is finite because B is. From (v) and the properties of *Supp*, we have **(v')** $B' \cap \mathit{Supp}\ (t[\sigma]) = \emptyset$.

Note that $\mathit{Psupp}\ p \cup \mathit{Supp}\ (t[\sigma])$ is finite because both $\mathit{Psupp}\ p$ and $\mathit{Supp}\ (t[\sigma])$ are finite. With the finiteness of B' and (v'), we obtain the existence of $\tau \in \mathit{Perm}$ such that

(vi) $\mathit{Im}\ \tau\ B' \cap (\mathit{Psupp}\ p \cup \mathit{Supp}\ (t[\sigma])) = \emptyset$ and **(vii)** $\forall x \in \mathit{Supp}\ (t[\sigma]).\ \tau\ x = x$.

Let $\delta = \tau \circ \sigma$. By the functoriality of $\_[\_]$, we have $t[\delta] = t[\sigma][\tau]$. Also, from (vii) and the properties of *Supp*, we have $t[\sigma][\tau] = t[\sigma]$. Hence **(viii)** $t[\delta] = t[\sigma]$. Note also that, by the definitions of δ and B' we have **(ix)** $\mathit{Im}\ \delta\ B = \mathit{Im}\ \tau\ B'$.

From (iv) and the monotonicity of G, we have $G\ (\lambda t'.\ I'_G\ t' \wedge (\forall p' \in P.\ \varphi\ p'\ (t'[\delta])))\ B\ t$. Hence, by G's monotonicity and I'_G's equivariance, $G\ (\lambda t'.\ I'_G\ (t'[\delta]) \wedge (\forall p' \in P.\ \varphi\ p'\ (t'[\delta])))\ B\ t$. Hence, by G's equivariance, $G\ (\lambda t'.\ I'_G\ (t'[\delta^{-1}][\delta]) \wedge (\forall p' \in P.\ \varphi\ p'\ (t'[\delta^{-1}][\delta])))\ (\mathit{Im}\ \delta\ B)\ (t[\delta])$. Hence, by $\_[\_]$'s functoriality and $\delta \circ \delta^{-1} = 1_{\mathit{Var}}$, $G\ (\lambda t'.\ I'_G\ t' \wedge (\forall p' \in P.\ \varphi\ p'\ t'))\ (\mathit{Im}\ \delta\ B)\ (t[\delta])$. Hence, using (viii) and (ix), $G\ (\lambda t'.\ I'_G\ t' \wedge (\forall p' \in P.\ \varphi\ p'\ t'))\ (\mathit{Im}\ \tau\ B')\ (t[\sigma])$. From this, (vi) and (5), we get $\varphi\ p\ (t[\sigma])$, as desired. □

5 THE MOTIVATING EXAMPLE REVISITED

Thm. 7 generalizes Prop. 2, and is in relation to Thm. 5 what Prop. 2 is in relation to Prop. 1, where β-reduction is generalized to an arbitrary inductively defined predicate I_G on a nominal set. Indeed, we obtain Prop. 2 by instantiating, in Thm. 7, $\mathcal{T}$ to the canonical nominal-set structure on LTerm^2 and $G : (\mathit{LTerm}^2 \to \mathit{Bool}) \to (\mathcal{P}_{\mathsf{fin}}(\mathit{Var}) \to \mathit{LTerm}^2 \to \mathit{Bool})$ to the operator described in Fig. 2.

Remark 8. In fact, instantiating Thm. 7 to the β-reduction relation does not give exactly Prop. 2 but a slight improvement of it, which in the (|Beta|) case also assumes $x \notin \mathit{FV}\ t_2$. Indeed, the assumption $B \cap (\mathit{Psupp}\ p \cup \mathit{Supp}^{\mathcal{T}} t) = \emptyset$ from Thm. 7 gives in the (|Beta|) case the assumption

$$G\ \varphi\ B\ (s, s') \iff \begin{array}{ll} (1) & (\exists x, t_1, t_2.\ B = \{x\} \land s = Lm\ x\ t_1 \land s' = t_1[t_2/x]) \lor \\ (2) & (\exists x, t, t'.\ \varphi\ (t, t') \land B = \{x\} \land s = Lm\ x\ t \land s' = Lm\ x\ t') \lor \\ (3) & (\exists t_1, t_2, t_1'.\ \varphi\ (t_1, t_1') \land B = \emptyset \land s = Ap\ t_1\ t_2 \land s' = Ap\ t_1'\ t_2) \lor \\ (4) & (\exists t_1, t_2, t_2'.\ \varphi\ (t_2, t_2') \land B = \emptyset \land s = Ap\ t_1\ t_2 \land s' = Ap\ t_1\ t_2') \end{array}$$

Fig. 2. The operator associated to λ-calculus β-reduction

$x \notin Psupp\ p \cup FV\ (Ap\ (Lm\ x\ t_1)\ t_2)) \cup FV\ (t_1[t_2/x])$, i.e., $x \notin Psupp\ p$ and $x \notin FV\ t_2$. Thus, we obtain as an extra hypothesis in the *induction* proof rule (making induction easier) exactly what Urban et al. must add as an extra hypothesis in the underlying *introduction* rule (making introduction harder).

Note that, for any inductive predicate (regardless of bindings) there is a"tension" between the introduction rules and the induction principle, namely by strengthening or weakening the hypotheses in the defining rules one becomes harder and the other easier to apply. From an abstract standpoint, what a strong induction principle achieves by taking advantage of the binding structure is to have the cake and eat it to, i.e., make induction easier to apply without affecting the introduction rules. This seems connected with the *some/any* principle from Nominal Logic [Gabbay and Pitts 2002, Prop. 3.4] [Pitts 2003, Prop. 4], which states that existential and universal quantification over fresh variables are equivalent (and forms the basis for the *freshness quantifier* [Gabbay and Pitts 2002]). Indeed, pushing this principle through the inductive definition while turning any (implicitly) existentially quantified fresh variables into universally quantified ones, under suitable assumptions about the definition could lead to an alternative proof of strong rule induction.

While the choice of $\mathcal{T}$ is straightforward, the choice of G requires some explanation. First note that, in Fig. 2's definition of G, everything but the treatment of the $B \in \mathcal{P}_{\mathsf{fin}}(Var)$ argument is completely determined by the original definition of the β-reduction predicate $\Rightarrow : LTerm \to LTerm \to Bool$ from Fig. 1. Indeed, if we ignore the treatment of B (the highlighted bits), we obtain the definition of a monotonic operator from $(LTerm^2 \to Bool) \to (LTerm^2 \to Bool)$ that, modulo currying, is exactly the operator underlying the definition of $\Rightarrow$ (as its least fixpoint, via Knaster–Tarski), where the four disjuncts from Fig. 2 correspond to the four rules from Fig. 1.

As for the B argument, its value is also completely determined by virtue of its role: *to store the bound variables that might occur in the conclusions of the rules.* In this case, we have at most one variable, so B will be either a singleton or the empty set. In general, variables may be bound within complex binding structures, e.g., nested record patterns as in the POPLmark Challenge 2B [Aydemir et al. 2005]. We are not interested in the exact form of these structures, but (at least for now) only in the set of variables that they contain.

Remark 9. Above, we argued that B is uniquely determined when we think of it as storing all the bound variables from a rule's conclusion. However, as one of the anonymous reviewers noted, an arbitrary $B \in \mathcal{P}_{\mathsf{fin}}(Var)$ above that minimal value would also work. In other words, we can loosen B upwards, i.e., in the definition of G from Fig. 2 replace the condition $B = \{x\}$ with $x \in B$ in disjuncts (1) and (2) and remove the condition $B = \emptyset$ form disjuncts (3) and (4) (since $\emptyset \subseteq B$ would be vacuous). All the needed checks, including $\mathcal{T}$-refreshability and equivariance, would also succeed for this looser definition of G.

Let us check that these choices of $\mathcal{T}$ and G satisfy the hypotheses of Thm. 7. G is obviously monotonic (as all logical connectives appearing in it are in the positive fragment of first-order logic). And G is equivariant because all operators appearing in it are equivariant.

It remains to check that G is $\mathcal{T}$-refreshable. To this end, let $\varphi : T \to Bool$ be an equivariant predicate, let $B \in \mathcal{P}_{\mathsf{fin}}(Var)$ and $(s, s') \in T = LTerm^2$, and assume $G\ \varphi\ B\ (s, s')$. We must find

$B' \in \mathcal{P}_{\text{fin}}(Var)$ such that $B' \cap Supp^{\mathcal{T}}(s, s') = \emptyset$, i.e., **(i)** $B' \cap (FV\, s \cup FV\, s') = \emptyset$, and **(ii)** $G\ \varphi\ B'\ (s, s')$. We distinguish four cases, depending on which disjunct from G's definition applies to (ii):

(1) Assume $B = \{x\}$, $s = Lm\ x\ t_1$ and $s' = t_1[t_2/x]$ for some x, t_1, t_2. We choose x' to be completely fresh (i.e., fresh for x, t_1 and t_2) and take $B' = \{x'\}$. Now, (i) holds by the choice of x'. Moreover, (ii) holds by virtue of the same disjunct (the first one) in the definition of G holding, with the existential witnesses $x', t_1[x' \leftrightarrow x], t_2$. Indeed:
- $B' = \{x'\}$ holds by definition;
- $s = Lm\ x\ t_1 = Lm\ x'\ (t_1[x' \leftrightarrow x])$ from properties of abstraction;
- $s' = t_1[t_2/x] = t_1[x' \leftrightarrow x][t_2/x']$ from properties of substitution.

(2) Assume $\varphi(t, t')$, $B = \{x\}$, $s = Lm\ x\ t$ and $s' = Lm\ x\ t'$ for some x, t, t'. Like before, we choose x' to be completely fresh and take $B' = \{x'\}$. Again, (i) holds by the choice of x', and (ii) holds by virtue of the same disjunct (the second one) in the definition of G, with the existential witnesses $x', t[x' \leftrightarrow x], t'[x' \leftrightarrow x]$:
- $\varphi\ (t[x' \leftrightarrow x], t'[x' \leftrightarrow x])$ because $\varphi(t, t')$ and φ is equivariant;
- $B' = \{x'\}$ holds by definition;
- $s = Lm\ x\ t = Lm\ x'\ (t[x' \leftrightarrow x])$ from properties of abstraction;
- $s' = Lm\ x\ t' = Lm\ x'\ (t'[x' \leftrightarrow x])$ from properties of abstraction.

(3) Assume $\varphi(t_1, t_1')$, $B = \emptyset$, $s = Ap\ t_1\ t_2$ and $s' = Ap\ t_1'\ t_2$ for some t_1, t_2, t_1'. Then (i) and (ii) hold trivially, taking $B' = \emptyset$ and, in (ii), using the same disjunct (the third one) with t_1, t_2, t_1' as witnesses.

(4) Similar to (3).

6 ON INSTANTIATING OUR THEOREM

As our examples suggest, our criterion is widely applicable. But in addition to the scope question, we are also interested in the formal engineering question on how difficult it is to instantiate this criterion. Fortunately, the instantiation follows well-understood patterns, facilitating automation.

Next, we will extrapolate from our §5 discussion about β-reduction, emphasizing the wider generality of the ideas presented there. The hypothetical scenario we consider is starting with a predicate over syntax with bindings specified inductively via rules, and wishing to deploy our Thm. 7 to obtain a strong rule induction principle for it—which in the case of β-reduction would be Prop. 2.

The operator G associated to our given predicate can be determined from its rules: G is a disjunction consisting of one disjunct for each rule; and each disjunct is an existential, quantifying over all component items (variables, terms, etc.) in the corresponding rule. This process, of extracting from a rule-based specification the underlying operator that ends up capturing the specified predicate as its least fixed point, is well-understood, and has been automated in several theorem provers, including the HOL-based provers HOL4 [HOL 2024; Gordon and Melham 1993], HOL Light [Harrison 2024] and Isabelle/HOL [Nipkow et al. 2002]. In addition, here we need to plug in the value of the set-of-bound-variables argument B, which in each disjunct of G is the set of variables bound (or substituted) in the conclusion of the corresponding rule. This requires knowing the involved binding structures, and can be facilitated by tools that track bindings at datatype-definition time—which include Nominal Isabelle [Urban 2008; Urban and Kaliszyk 2012] (and our own tool we describe in App. G).

Checking G's monotonicity and equivariance tends to be routine. Most HOL-based provers track monotonicity as part of their inductive definition facilities. Nominal Isabelle tracks equivariance based on its compositionality [Pitts 2013], and Isabelle's Lifting&Transfer tool [Huffman and Kunčar 2013] tracks the related notion of parametricity [Reynolds 1983; Wadler 1989].

The only check that could be non-trivial is that of the $\mathcal{T}$-refreshability of G, which means: We start with an equivariant $\varphi : T \to \mathit{Bool}$, a B and a $t \in T$ such that $G\ \varphi\ B\ t$ holds; and must produce a B' such that $B' \cap \mathit{Supp}^{\mathcal{T}} t = \emptyset$ and $G\ \varphi\ B'\ t$. As hinted in §5, this can proceed via the following heuristic:

Step 1: Because $G\ \varphi\ B\ t$ holds and is expressed as a disjunction of existentials, we obtain some items, usually terms or variables, that satisfy one of the disjuncts, which we will refer to as the "original" items (e.g., x, t, t' in the second disjunct of $G\ \varphi\ B\ t$ in Fig. 2).

Step 2: We pick some completely fresh variables to replace the variables in B, i.e., for each variable x in B we pick a fresh variable x', and take B' to be the corresponding disjoint copy of B (consisting of the "primed" variables). This ensures that $B' \cap \mathit{Supp}^{\mathcal{T}} t = \emptyset$ holds.

Step 3: To prove $G\ \varphi\ B'\ t$, which again is a disjunction of existentials, we prove the same disjunct as the one known to hold for $G\ \varphi\ B\ t$ (e.g., the second disjunct of $G\ \varphi\ B'\ t$ if it is the second disjunct of $G\ \varphi\ B\ t$ that happened to hold), and as witnesses for this disjunct's existentials we plug in the original items (that witnessed the corresponding disjunct of $G\ \varphi\ B\ t$) *in which we swap the original variables x with their fresh counterparts x' as appropriate.* Here, "as appropriate" means that swapping only takes place if the variable x is either equal to the considered original item, or that item is in the scope of its binding. Thus, for example, for Fig. 2's first disjunct we replace x with x' and t_1 with $t_1[x \leftrightarrow x']$, but t_2 stays as it is (because the latter is not in the scope of the bound variable x). It remains to verify the disjunct of $G\ \varphi\ B'\ t$ whose existentials have been instantiated with these witnesses. For example, in the case of Fig. 2's second disjunct, knowing that $\varphi(t, t')$, $B = \{x\}$, $s = \mathit{Lm}\ x\ t$ and $s = \mathit{Lm}\ x'\ t'$ hold, we want to verify that $\varphi(t[x \leftrightarrow x'], t'[x \leftrightarrow x'])$, $B' = \{x'\}$, $s = \mathit{Lm}\ x'\ (t[x \leftrightarrow x'])$ and $s = \mathit{Lm}\ x'\ (t'[x \leftrightarrow x'])$ also hold. Among these goals to be proved:

- those involving B' follow from the prior knowledge about B and the construction of B' (e.g., $B' = \{x'\}$ follows from $B = \{x\}$);
- those involving the occurrences of the predicate, e.g., $\varphi(t[x \leftrightarrow x'], t'[x \leftrightarrow x'])$, follow from the corresponding fact in the original disjunct, e.g., $\varphi(t, t')$, and the assumed equivariance of φ.

As for the other goals, such as $s = \mathit{Lm}\ x\ t$ implying $s = \mathit{Lm}\ x'\ (t[x \leftrightarrow x'])$, which amounts to $\mathit{Lm}\ x\ t = \mathit{Lm}\ x'\ (t[x \leftrightarrow x'])$, they say that the original items can be replaced in certain contexts by the "refreshed" (swapped) items. For these, we have no general recipe but an empirical observation validated on many examples: These goals tend to be reducible to standard properties of the syntactic operators (constructors, swapping, substitution, etc.).

Remark 10. A crucial part of the above heuristic for checking $\mathcal{T}$-refreshability is assuming that the predicate argument φ of G is equivariant and holds for some "original" items, and wanting to prove that it holds for modifications of these items where the variables from B are swapped "as appropriate", i.e., swapped or not depending on their being in the scope of bindings in the rules' conclusions. (Note that φ intuitively stands for the inductively defined predicate during iteration through G.) Favorable situations that work out of the box are when, in the hypotheses of each defining rule, each occurrence of the inductively defined predicate is: **(A)** either applied to items that are *all not in* the scope of bound variables in the conclusion, yielding trivial goals such as "$\varphi(t, t')$ implies $\varphi(t, t')$", or **(B)** applied to items that are *all in* the scope of bound variables in the conclusion, yielding goals such as "$\varphi(t, t')$ implies $\varphi(t[x \leftrightarrow x'], t'[x \leftrightarrow x'])$" which follow from φ's equivariance. Otherwise, we encounter a hybrid situation, i.e., an in-hypotheses occurrence of the inductively defined predicate that is **(C)** applied to some items in, and to some items not in the scope of bound variables in the conclusion. Then, we end up with hybrid goals such as "$\varphi(t, t')$ implies $\varphi(t[x \leftrightarrow x'], t')$". In these cases, the only way forward is if the given rule guarantees, perhaps via a side-condition, the freshness of the original variables for the offending original terms (i.e., those subjected to swapping), e.g., the freshness of x for t—because, x' being fresh as well, we would have $t[x \leftrightarrow x'] = t$ so we could fall back on case (A).

Remark 11. Let us see what problems we would incur with our β-reduction example if we tried to check that G satisfies not $\mathcal{T}$-refreshability but the stronger condition that we called $\mathcal{T}$-freshness (in Def. 6). The latter requires that $G\ \varphi\ B\ t$ implies $B \cap Supp^{\mathcal{T}} t = \emptyset$, i.e., that $B \cap Supp^{\mathcal{T}} t = \emptyset$ follows from each disjunct in the definition of $G\ \varphi\ B\ t$, corresponding to a rule in the inductive definition of β-reduction. So we want all the variables in B, i.e., those appearing bound in the rule's conclusion, to be prevented from (also) appearing free in the rule's conclusion. This works for all the rules except the first one in Fig. 1 (corresponding to the first disjunct in Fig. 2), where x which appears bound in the conclusion is not prevented from also appearing free in the conclusion, e.g., within t_2. Thus, a fix to get $\mathcal{T}$-freshness would be adding the side-condition that x be fresh for t_2, as seen in the rule (Beta') from §2; and a similar situation occurs with the "Parallel Beta" rule (ParBeta) from §2, which to validate $\mathcal{T}$-freshness must become (ParBeta'). In fact, as detailed in App. A, our $\mathcal{T}$-freshness generalizes Urban et al. [2007]'s criterion. The next section further explores the difference between the two criteria.

7 MORE ON THE COMPARISON WITH THE URBAN ET AL. CRITERION

As stated at the end §2, our Thm. 7 improves on Urban et al. [2007]'s result in two ways: (1) not necessitating the addition of side-conditions to capture concrete systems and (2) going beyond syntactic format for the rules. In this section, taking advantage of the availability of more concepts and notation, we will further illustrate what improvement (1) amounts to by going into finer details.

We consider again the standard β-reduction relation described in Fig. 1. So our theorem applies to this relations' definition as is, whereas in order to apply Urban et al.'s criterion (Theorem 1 from [Urban et al. 2007]) one requires a modification of the definition, namely the addition of the side-condition $x \notin FV\ \ t_2$ to the (Beta) rule, i.e., the replacement of (Beta) with the rule (Beta') shown below:

$$Ap\ (Lm\ x\ t_1)\ t_2 \Rightarrow t_1[t_2/x]\ \text{(Beta')}\ [x \notin FV\ \ t_2]$$

It turns out that the modified system can be proved equivalent with (equal to) the original one—and Urban et al. noted that this tends to be the case in concrete examples, but they left open the problem of proving that in a general setting (such as the setting, based on a format for schematic rules).

Let us see how to prove that the above two concrete systems, the original one and the one modified by having (Beta') replacing (Beta), are equivalent. Clearly the original one is at least as strong as the modified one. Conversely, to prove that the modified one is at least as strong as the original one, we essentially need to prove that (Beta) can be "simulated" by (Beta'). And indeed, this intuitively seems to be the case because the bound variable x in (Beta) can in principle be renamed to something fresh for t_2, and this renaming should be immaterial (since terms are quotiented to α-equivalence). However, we cannot simply invoke such a renaming without further argumentation. This is because, in (Beta) and (Beta'), the term t_1 appears not only inside the scope of a λ-bound x (within $Lm\ x\ t_1$) by also outside this scope (within $t_1[2_2/x]$)—so while the first occurrence of t_1 has x bound, the second occurrence "exposes" the name x. We must take this into account when doing the inference of (Beta) from (Beta'), which goes as follows: We assume (Beta') holds. To prove (Beta), let x be a variable and t_1, t_2 be terms; we need to show $Ap\,(Lm\ x\ t_1)\ t_2 \Rightarrow t_1[t_2/x]$. To this end, we pick a completely fresh variable, say x', and define t_1' by swapping (or alternatively substituting) x with x' in t_1, namely $t_1' = t_1[x \leftrightarrow x']$. Then using the properties of swapping and substitution and the fact that x' is fresh for t_1, we obtain that $t_1'[t_2/x'] = t_1[t_2/x]$; and using the properties of λ-abstraction (stemming from α-equivalence), we obtain that $Lm\ x'\ t_1' = Lm\ x\ t_1$. This allows us to infer the desired instance of (Beta), namely $(Lm\ x\ t_1)\ t_2 \Rightarrow t_1[t_2/x]$, from an instance of (Beta'), namely $(Lm\ x'\ t_1')\ t_2 \Rightarrow t_1'[t_2/x']$; the latter is indeed an instance of (Beta') since x' is fresh for t_2.

$$t \Longrightarrow t \text{ (Refl)} \qquad \frac{t \Longrightarrow t'}{Lm\ x\ t \Longrightarrow Lm\ x\ t'} \text{ (Xi)}$$

$$\frac{t_1 \Longrightarrow t_1' \qquad t_2 \Longrightarrow t_2'}{Ap\ t_1\ t_2 \Longrightarrow Ap\ t_1'\ t_2'} \text{ (Ap)} \qquad \frac{t_1 \Longrightarrow t_1' \qquad t_2 \Longrightarrow t_2'}{Ap\ (Lm\ x\ t_1)\ t_2 \Longrightarrow t_1'[t_2'/x]} \text{ (ParBeta)}$$

Fig. 3. λ-calculus parallel β-reduction

In the case of (Beta) versus (Beta'), the technicalities were relatively simple thanks to dealing with an axiom (i.e., a rule with no hypotheses), but when dealing with proper rules (as is more often the case) inference is more difficult. We illustrate this with the parallel β-reduction relation briefly mentioned in §2, which was Urban et al.'s initial motivating example. Its definition is shown in Fig. 3. Again, our theorem applies to this definition as is, whereas Urban et al.'s theorem requires the addition of the side-condition $x \notin FV\ t_2 \cup FV\ t_2']$ to the (ParBeta) rule, i.e., the replacement of (ParBeta) with the rule (ParBeta') shown below:

$$\frac{t_1 \Longrightarrow t_1' \qquad t_2 \Longrightarrow t_2'}{Ap\ (Lm\ x\ t_1)\ t_2 \Longrightarrow t_1'[t_2'/x]} \ \begin{array}{l}\text{(ParBeta')}\\ [x \notin FV\ t_2 \cup FV\ t_2']\end{array}$$

Now, it is not even true that, in isolation (that is, regardless of what the other rules of the system are) the rule (ParBeta) is inferable from the rule (ParBeta'). What *is* inferable from (ParBeta'), applying an argument similar to the one sketched above for (Beta) versus (Beta') (that is, picking a fresh x' and using properties of substitution, swapping and constructors), is only a modification of (ParBeta) that replaces the hypotheses $t_1 \Longrightarrow t_1'$ and $t_2 \Longrightarrow t_2'$ with $t_1[x \leftrightarrow x'] \Longrightarrow t_1'[x \leftrightarrow x']$ and $t_2[x \leftrightarrow x'] \Longrightarrow t_2'[x \leftrightarrow x']$ for some fresh x'. Then, after we prove equivariance for the entire system featuring (ParBeta') and the other rules (so depending on the well-behavedness of the other rules as well), we can replace the modified hypotheses with the original hypotheses of (ParBeta)—concluding the proof that the two versions are equivalent (since the opposite direction, i.e., moving from (ParBeta) to (ParBeta'), is again trivial).

Note that the above arguments for getting rid of certain side-conditions involved an equivariance proof, and also some specific properties of the operators participating in the rules, such as substitution. Our strong rule induction criterion, Thm. 7, can be regarded as providing a generalization of such arguments baked into the argument for the soundness of strong rule induction.

A final note about the above rule (ParBeta'): The $x \notin FV\ t_2'$ part of the added side-condition is seen to be redundant also because parallel β-reduction can be proved to not any new free variables (when moving from left to right), so $x \notin FV\ t_2'$ follows from $x \notin FV\ t_2$. But general-purpose criteria such as Urban et al.'s and ours are not addressing such specific semantic properties though (nor do they assume, of course, that the defined predicate takes the form of a transition relation). In particular, while our criterion does not require the addition of side-conditions, it does not provide a mechanism for detecting redundant side-conditions when already part of the original rules.

Overview of the Next Two Sections. In what follows, we validate, challenge and refine the metatheory through examples that exhibit more complexity than the λ-calculus along several directions: scope extrusion and complex side-conditions (π-calculus, §8.1), environments (System $F_{<:}$, §8.2), and terms with infinitely many variables (infinitary FOL §9.1 and λ-calculus §9.3). While the π-calculus example showcases the improvements of our criterion over the state of the art, we chose to present the other examples because they have challenged this criterion, inspiring further improvements and generalizations: making inductive information available for refreshability (§8.3), allowing infinitary structures (§9.2), and considering binders explicitly while loosening equivariance (§9.4).

8 MORE EXAMPLES AND REFINEMENTS

For the syntaxes in this section, we will implicitly use standard notions such as permutation and free variables. They all form nominal sets similarly to how the λ-calculus syntax does.

8.1 Example: the π-calculus

In this subsection, variables will sometimes be called "names" or "channels". We also let a, b (in addition to x, y, z) range over variables. The set *Proc*, of π-calculus *processes* [Milner 1999; Milner et al. 1992], ranged over by P, Q, R etc., is described by a grammar of the following form (where we omit the constructors that will play no role in our discussion):

$$P \quad ::= \quad \ldots \mid P \parallel Q \mid !P \mid \overline{a}\, x.\, P \mid a(x).\, P \mid \nu(x).\, P$$

We assume that x is bound in P within processes of the form $a(x).\, P$ and $\nu(x).\, P$; and processes are equated modulo the induced alpha-equivalence. The shown constructors are, in order: parallel composition, replication, output (of name x on the channel a), input (of a generic name x on channel a), and restriction/hiding (of the name x).

The set *Act* of *actions*, ranged over by α, is given by the grammar:

$$\alpha \ ::= \ \tau \mid a\, x \mid \overline{a}\, x \mid a(x) \mid \overline{a}(x)$$

The above are, in order: the silent action, the input of a (free) name x on channel a, the output of a (free) name x on channel a, the symbolic input of a (bound) name x on channel a, and the output of a bound name x on channel a. The first three types will be called *free actions*; we let *fra* α express the fact that α is a free action.

We let *ns* α, the set of names of an action α, consist of all the names appearing in that action (so *ns* α is empty if $\alpha = \tau$ and otherwise it has at most two elements). We also let *bns* α, the set of *bound names of* α, be $\{x\}$ if α has the form $a(x)$ or $\overline{a}(x)$, and $\emptyset$ otherwise. And *fns* α, the set of *free names of* α, be $\{a\}$ if α has the form $a(x)$ or $\overline{a}(x)$, and *ns* α otherwise. In particular, we have *ns* α = *bns* $\alpha \cup$ *fns* α, though *bns* α and *fns* α may not be disjoint.

A process can take an action by consuming one of its communicating prefixes ($\overline{a}\, x.$ or $a(x).$) and transitioning to a remainder process. This is described by an inductively defined transition relation, using rules including ones shown in Fig. 4.

There are two main variants of operational semantics for the π-calculus—early-instantiation and late-instantiation—depending on whether input instantiation is exhibited "early" for single processes or "late" during communication. Fig. 4 shows the binding-interesting rules for both variants. (For conciseness, we used a notion of action that is broad enough to accommodate both variants.)

A binding behavior characteristic to the π-calculus is *scope extrusion*: Via the rule (Open), a process, say $\nu(x).\, Q$, "opens" the scope of a previously bound variable x; then, via the rule (CloseLeftE) or (CloseLeftL) (or their symmetrics), the scope is "closed" after another process P receives this bound name. At the end of this scope opening and closing session, a name x that was previously known to the process Q alone has now been shared with P—becoming a shared secret between the remainder processes P' and Q'.

Remark 12. In a naive formalization of the transition relation, namely as a *ternary* relation, a rule such as (Open) is known to be problematic for formal reasoning, essentially because it is resistant to strong induction [Bengtson 2010]. In fact, we can explain this problem in terms of our §6 heuristic for proving $\mathcal{T}$-refreshability. We would get stuck along the lines sketched in Remark 10: attempting to prove, for an equivariant predicate $\varphi : Proc \times Act \times Proc \longrightarrow Bool$ and a fresh x', the hopeless goal "$\varphi\,(P, \overline{a}\, x, P')$ implies $\varphi\,(P[x \leftrightarrow x'], \overline{a}\, x', P')$" while knowing that $a \neq x$ but *not* that x is fresh for P. (And while the "fix" of adding to (Open) the side-condition that x be fresh for P would indeed enable strong induction, it would also destroy the intended semantics by preventing P from sending any of

$$a(x).\,P \xrightarrow{a\,y} P[y/x] \ \text{(InpE)} \qquad \frac{P \xrightarrow{a\,x} P' \quad Q \xrightarrow{\overline{a}\,x} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \ \text{(ComLeftE)} \qquad \frac{P \xrightarrow{a\,x} P' \quad Q \xrightarrow{\overline{a}(x)} Q'}{P \parallel Q \xrightarrow{\tau} \nu(x).\,(P' \parallel Q')} \ \begin{array}{l}\text{(CloseLeftE)}\\ [x \notin \{a\} \cup FV\,P]\end{array}$$

Rules specific to the early-instantiation semantics

$$a(x).\,P \xrightarrow{a(x)} P \ \text{(InpL)} \qquad \frac{P \xrightarrow{a(x)} P' \qquad Q \xrightarrow{\overline{a}\,y} Q'}{P \parallel Q \xrightarrow{\tau} P'[y/x] \parallel Q'} \ \text{(ComLeftL)} \qquad \frac{P \xrightarrow{a(x)} P' \qquad Q \xrightarrow{\overline{a}(x)} Q'}{P \parallel Q \xrightarrow{\tau} \nu(x).\,(P' \parallel Q')} \ \text{(CloseLeftL)}$$

Rules specific to the late-instantiation semantics

$$\frac{P \xrightarrow{\overline{a}\,x} P'}{\nu(x).\,P \xrightarrow{\overline{a}(x)} P'} \ \begin{array}{l}\text{(Open)}\\ [a \neq x]\end{array} \qquad \frac{P \xrightarrow{\alpha} P'}{\nu(x).\,P \xrightarrow{\alpha} \nu(x).\,P'} \ \begin{array}{l}\text{(ScopeFree)}\\ [\mathit{fra}\ \alpha,\ x \notin \mathit{ns}\ \alpha]\end{array}$$

$$\frac{P \xrightarrow{\overline{a}(x)} P'}{\nu(y).\,P \xrightarrow{\overline{a}(x)} \nu(y).\,P'} \ \begin{array}{l}\text{(ScopeBound)}\\ [y \notin \{a, x\},\ x \notin FV\ P \cup \{a\}]\end{array} \qquad \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \ \begin{array}{l}\text{(ParLeft)}\\ [\mathit{bns}\ \alpha \cap FV(P, Q) = \emptyset]\end{array}$$

Rules common to both styles of semantics

Fig. 4. π-calculus transition relation

$$a(x).\,P \xrightarrow{a\,y} P[y/x] \ \begin{array}{l}\text{(InpE')}\\ [x \notin \{a, y\}]\end{array} \qquad \frac{P \xrightarrow{a\,x} P' \qquad Q \xrightarrow{\overline{a}(x)} Q'}{P \parallel Q \xrightarrow{\tau} \nu(x).(P' \parallel Q')} \ \begin{array}{l}\text{(CloseLeftE')}\\ [x \notin \{a\} \cup FV(P, Q)]\end{array}$$

$$\frac{P \xrightarrow{a(x)} P' \qquad Q \xrightarrow{\overline{a}\,y} Q'}{P \parallel Q \xrightarrow{\tau} P'[y/x] \parallel Q'} \ \begin{array}{l}\text{(ComLeftL')}\\ [x \notin FV(P, Q, Q')]\end{array}$$

$$\frac{P \xrightarrow{a(x)} P' \qquad Q \xrightarrow{\overline{a}(x)} Q'}{P \parallel Q \xrightarrow{\tau} \nu(x).\,(P' \parallel Q')} \ \begin{array}{l}\text{(CloseLeftL')}\\ [x \notin FV(P, Q)]\end{array} \qquad \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \ \begin{array}{l}\text{(ParLeft')}\\ [\mathit{bns}\ \alpha \cap FV(P, Q) = \emptyset,\ \mathit{bns}\ \alpha \cap \mathit{fns}\ \alpha = \emptyset]\end{array}$$

Fig. 5. π-calculus transitions augmented to accommodate prior state-of-the-art strong rule induction

its known (i.e., free) names.) This is not a problem with our criterion, but a situation where applying Barendregt's convention would be unsound; a similar example is given Urban et al. [2007, p.38].

An elegant solution to the above problem comes from noting the following about the intended semantics: that any name which is bound in the action labeling the transition, e.g., a name x sent via an $\overline{a}(x)$ action, has its identity "hidden"; in particular, until further extruding actions, is unavailable to any other process besides the one that sends it and the one that receives it. This is best modeled syntactically by assuming that such a name x *gets bound from within the action into the remainder process*. Thus, in the conclusion $\nu(x).\,P \xrightarrow{\overline{a}(x)} P'$ of (Open), we think of the occurrence of x in $\overline{a}(x)$ as binding any (free) occurrence of x in P'. This solution was pursued in his thesis by Bengtson [Bengtson 2010], who (crediting Milner et al. for the idea [Milner 1993; Milner et al. 1992]) formalizes the π-calculus transition relation not as a ternary relation between a source process, an action and a target process, but as a binary relation between a source process and a commitment, the latter being a pair (action, remainder process) up to alpha-equivalence.

Following Bengtson, we thus define the set *Com* of *commitments* to consist of pairs $C = (\alpha, P)$ up to alpha-equivalence. That is, commitments are generated by the (nonrecursive) grammar

$$C \ ::= \ (\tau, P) \ \mid \ (a\,x, P) \ \mid \ (\overline{a}\,x, P) \ \mid \ (a(x), P) \ \mid \ (\overline{a}(x), P)$$

(having one production for each action type) with the assumption that, in a commitment of the form $(a(x), P)$ or $(\overline{a}(x), P)$, x is bound in P; and commitments are identified modulo alpha-equivalence.

Under this commitment-based view, Fig. 4 stays the same, but now we read $P \xrightarrow{\alpha} P'$ as notation for $tran\ P\ (\alpha, P')$, where $tran : Proc \to Com \to Bool$ is a (binary) relation between $Proc$ and Com. Thus, the rules in Fig. 4 give an inductive definition of $tran$. In this setting, the problem with (Open) from Remark 12 vanishes, because now both occurrences of x from its conclusion are bound: one binding in P and one in P'. Hence our heuristic for $\mathcal{T}$-refreshability succeeds, as it just needs to check "$\varphi\ P\ (\overline{a}\,x)\ P'$ implies $\varphi\ (P[x \leftrightarrow x'])\ (\overline{a}\,x')\ (P'[x \leftrightarrow x'])$", an instance of φ's equivariance.

And indeed, applying Thm. 7 to the inductive rules from Fig. 4, we obtain the desired strong rule induction, where in the inductive hypotheses we can assume that all the bound variables referenced in these rules are fresh for the parameters. For example, here is the strong rule induction we obtain if we consider that the transition relation is defined by a particular selection of the Fig. 4 rules, namely (InpE), (CloseLeftE) and (ComLeftL), (CloseLeftL) and (ParLeft):

Prop 13. Let $(P, Psupp : P \to \mathcal{P}_{\mathsf{fin}}(Var))$ be a parameter structure. Let $\varphi : P \to Proc \to Com \to Bool$ and assume the following hold:

- ⦇InpE⦈: $\forall p, a, x, y, P, Q.\ x \notin Psupp\ p \wedge x \notin \{a, y\} \longrightarrow \varphi\ p\ (a(x).P)\ (\overline{a}\,y, P[y/x])$
- ⦇CloseLeftE⦈: $\forall p, a, x, P, P', Q, Q'.\ x \notin Psupp\ p \wedge x \notin FV\ Q\ \wedge x \neq a \wedge x \notin FV\ P\ \wedge$
  $(P \xrightarrow{a\,x} P') \wedge (\forall q.\ \varphi\ q\ P\ (a\,x, P')) \wedge\ (Q \xrightarrow{\overline{a}(x)} Q') \wedge (\forall q.\ \varphi\ q\ Q\ (\overline{a}(x), Q')) \longrightarrow$
  $\varphi\ p\ (P \parallel Q)\ (\tau, P' \parallel Q')$
- ⦇ComLeftL⦈: $\forall p, a, x, y, P, P', Q, Q'.\ x \notin Psupp\ p \wedge x \notin FV\ (P, Q, Q') \wedge$
  $(P \xrightarrow{a(x)} P') \wedge (\forall q.\ \varphi\ q\ P\ (a(x), P')) \wedge (Q \xrightarrow{\overline{a}\,y} Q') \wedge (\forall q.\ \varphi\ q\ Q\ (\overline{a}\,y, Q')) \longrightarrow$
  $\varphi\ p\ (P \parallel Q)\ (\tau, P'[y/x] \parallel Q')$
- ⦇CloseLeftL⦈: $\forall p, a, x, P, P', Q, Q'.\ x \notin Psupp\ p \wedge x \notin FV\ (P, Q) \wedge$
  $(P \xrightarrow{a(x)} P') \wedge (\forall q.\ \varphi\ q\ P\ (a(x), P')) \wedge (Q \xrightarrow{\overline{a}(x)} Q') \wedge (\forall q.\ \varphi\ q\ Q\ (\overline{a}(x), Q')) \longrightarrow$
  $\varphi\ p\ (P \parallel Q)\ (\tau, P' \parallel Q')$
- ⦇ParLeft⦈: $\forall p, \alpha, P, P', Q.\ bns\ \alpha \cap Psupp\ p = \emptyset \wedge bns\ \alpha \cap fns\ \alpha = \emptyset \wedge bns\ \alpha \cap FV(P, Q) = \emptyset\ \wedge$
  $(P \xrightarrow{\alpha} P') \wedge (\forall q.\ \varphi\ q\ P\ (\alpha, P'))\ \longrightarrow\ \varphi\ p\ (P \parallel Q)\ (\alpha, P' \parallel Q)$

Then $\forall p, P, \alpha, P'.\ (P \xrightarrow{\alpha} P') \longrightarrow \varphi\ p\ P\ (\alpha, P')$.

On the other hand, using the state of the art [Urban et al. 2007] as implemented in Nominal Isabelle (which Bengtson used in his formalization [Bengtson 2012]), to get the same result one needs to augment the rules with side-conditions as highlighted in Fig. 5. These would ensure that the system satisfies not only $\mathcal{T}$-refreshability, but also $\mathcal{T}$-freshness. Indeed, as we discussed in Remark 11, $\mathcal{T}$-freshness in concrete examples amounts to the variables appearing bound in the conclusion of a rule being prevented from also appearing free in that conclusion. For example, $\mathcal{T}$-freshness does not hold for the rule (CloseLeftL) from Fig. 4 because x, which appears bound in the conclusion, can also appear free there, namely within P and Q—so to make $\mathcal{T}$-freshness hold one must add the side-condition highlighted in (CloseLeftL') from Fig. 5. Unlike in the situation from Remark 12, and like in those from Remark 11, here these fixes (required for $\mathcal{T}$-freshness but not for $\mathcal{T}$-refreshability) do not destroy the intended meaning of the definitions, but introduce unnecessary clutter.

Some versions of π-calculus [Sangiorgi and Walker 2001] distinguish between structural and operational rules—they too admit strong rule induction (as we illustrate on an example in App. B).

$$\frac{wf\ \Gamma\quad FV\ S \subseteq dom\ \Gamma}{\Gamma \vdash S <: Top}\ (\text{Top}) \qquad \frac{wf\ \Gamma \quad X \in dom\ \Gamma}{\Gamma \vdash (TVr\ X) <: (TVr\ X)}\ (\text{Refl-TV}) \qquad \frac{X <: S \in \Gamma \quad \Gamma \vdash S <: T}{\Gamma \vdash S <: T}\ (\text{Trans-TV})$$

$$\frac{\Gamma \vdash T_1 <: S_1 \qquad \Gamma \vdash S_2 <: T_2}{\Gamma \vdash (S_1 \to S_2) <: (T_1 \to T_2)}\ (\text{Arrow}) \qquad \frac{\Gamma \vdash T_1 <: S_1 \qquad \Gamma, X <: T_1 \vdash S_2 <: T_2}{\Gamma \vdash (\forall X <: S_1.\, S_2) <: (\forall X <: T_1.\, T_2)}\ (\text{All})$$

Fig. 6. System $F_{<:}$ subtyping

$$\begin{aligned}
&G\ \varphi\ B\ (\Gamma, S', T') \iff \\
&(\exists S.\ B = \emptyset \wedge S' = S \wedge T' = Top) \vee (\exists X.\ B = \emptyset \wedge S' = TVr\ X \wedge T' = TVr\ X) \vee \\
&(\exists X, Y, T.\ X <: Y \in \Gamma \wedge \varphi\ (\Gamma, TVr\ Y, T) \wedge B = \emptyset \wedge S' = TVr\ X \wedge T' = T) \vee \\
&(\exists S_1, S_2, T_1, T_2.\ \varphi\ (\Gamma, T_1, S_1) \wedge \varphi\ (\Gamma, T_2, S_2) \wedge B = \emptyset \wedge S' = (S_1 \to S_2) \wedge T' = (T_1 \to T_2)) \vee \\
&(\exists X, S_1, S_2, T_1, T_2.\ \varphi\ (\Gamma, T_1, S_1) \wedge \varphi\ ((\Gamma, X <: T_1), S_2, T_2) \wedge B = \{X\} \wedge S' = (\forall X <: S_1.\, S_2) \wedge T' = (\forall X <: T_1.\, T_2))
\end{aligned}$$

Fig. 7. The operator associated to System $F_{<:}$ subtyping

8.2 Example: System $F_{<:}$ subtyping

Next we look at the subtyping relation for System $F_{<:}$ [Aydemir et al. 2005; Cardelli et al. 1994; Curien and Ghelli 1992], an example combining type bindings with environment bindings.

In this subsection, the variables in *Var* will stand for type variables, and X, Y, Z etc. will range over them. The set *Type* of *types*, ranged over by S, T etc., is generated by the following grammar:

$$T \ ::=\ TVr\ X \ \mid\ Top \ \mid\ T \to S \ \mid\ \forall X <: T.\, S$$

So a type is either a (type) variable, or the maximum type *Top*, or a function type, or a universal type. We assume that, in a universal type $\forall X <: T.\, S$, the variable X is bound in S (but not in T); and types are equated modulo the induced notion of alpha-equivalence.

A *(typing) environment* Γ is a list of pairs variable-type, (X, T), written $X <: T$. *Env* denotes the set of environments. The domain $dom\ \Gamma$ of an environment consists of all the variables X for which some $X <: T$ is in Γ. An environment is said to be *well-formed*, written $wf\ \Gamma$, if whenever Γ has the form $\Gamma', X <: T, \Gamma''$, we have that $X \notin dom\ \Gamma'$ and $FV\ T \subseteq dom\ \Gamma'$—i.e., thinking of the environment as growing left-to-right with pairs, any new pair $X <: T$ must be such that X is fresh and T does not bring new (free) variables. *Subtyping* is a ternary relation between environments, types and types, written $\Gamma \vdash S <: T$, defined inductively in Fig. 6. On the way to instantiating Thm. 7 to this system, we obtain the operator G shown in Fig. 7. Thm. 7's conclusion would give us the following induction principle, avoiding parameter variables in the (|All|) case:

Prop 14. Let $(P, Psupp : P \to \mathcal{P}_{\text{fin}}(Var))$ be a parameter structure. Let $\varphi : P \to Env \to Type \to Type \to Bool$ and assume that:

- [cases different from (|All|) omitted, as they don't involve binders]
- (|All|): $\forall p, X, S_1, S_2, T_1, T_2.\ X \notin Psupp\ p \wedge X \notin FV(\Gamma, S_1, T_1) \wedge$
  $\Gamma \vdash T_1 <: S_1 \wedge (\forall q.\ \varphi\ q\ \Gamma\ T_1\ S_1) \wedge \Gamma, X <: T_1 \vdash S_2 <: T_2 \wedge (\forall q.\ \varphi\ q\ (\Gamma, X <: T_1)\ S_2\ T_2) \longrightarrow$
  $\varphi\ p\ \Gamma\ (\forall X <: S_1.\, S_2)\ (\forall X <: T_1.\, T_2)$

Then $\forall p, \Gamma, S, T.\ \Gamma \vdash S <: T \longrightarrow \varphi\ p\ \Gamma\ S\ T$.

However, when attempting to check Thm. 7's hypotheses, we get stuck at $\mathcal{T}$-refreshability. Namely, when deploying the heuristic sketched in §6, we encounter a problem with Fig. 6's (All) rule, i.e., with the fifth disjunct in Fig. 7's definition of G. While focusing on the second hypothesis of the (All) rule, for an equivariant $\varphi : Env \times Type \times Type \to Bool$, we know that **(i)** X' is fresh and **(ii)** $\varphi\ ((\Gamma, X <: T_1),\ S_2,\ T_2)$, and want to prove **(iii)** $\varphi\ ((\Gamma, X' <: T_1),\ S_2[X \leftrightarrow X'],\ T_2[X \leftrightarrow X'])$. (Note that, in (iii), Γ and T_1 are not subject to swapping, because in (All)'s conclusion they are not in the scope of X's binding.) However, φ's equivariance and (ii) only ensure

(iii') $\varphi\ ((\Gamma[X \leftrightarrow X'], X' <: T_1[X \leftrightarrow X']), S_2[X \leftrightarrow X'], T_2[X \leftrightarrow X'])$,

i.e., the variation of (iii) where swapping *is* applied to Γ and T_1. In short, we fall under one of those "hybrid" situations (case (C)) discussed in Remark 10. Since X' is fresh, we would have $\Gamma[X \leftrightarrow X'] = \Gamma$ and $T_1[X \leftrightarrow X'] = T_1$, hence (iii) would follow from (iii') and the problem would be solved, but only provided that: **(iv)** X is also fresh for Γ and T_1.

But currently there is no way to prove (iv)—which is a shame, because we *can* prove that $\Gamma, X <: T_1 \vdash S_2 <: T_2$ implies (iv), namely as follows: First, we prove by standard induction that, for all Γ', S, T, $\Gamma' \vdash S <: T$ implies $\mathit{wf}\ \Gamma'$. So $\Gamma, X <: T_1 \vdash S_2 <: T_2$ implies $\mathit{wf}\ (\Gamma, X <: T_1)$, which by the definition of wf implies that X is fresh for Γ and $FV\ T \subseteq FV\ \Gamma$, ultimately implying that X is fresh for T as well.

8.3 An inductively strengthened criterion

Thus, we would solve the problem if we could take advantage of properties of the inductively defined predicate when checking the conclusion of the $\mathcal{T}$-refreshability condition. Stepping back into §4's general setting, we are led to a weakening of $\mathcal{T}$-refreshability (highlighting the difference from our original definition, part of Def. 6):

Def 15. Given a nominal set $\mathcal{T} = (T, \_[]^{\mathcal{T}})$ and an operator $G : (T \to Bool) \to (\mathcal{P}_{\mathsf{fin}}(Var) \to T \to Bool)$, we say that G is *weakly $\mathcal{T}$-refreshable* when, for all $\varphi : T \to Bool$ such that $\forall t \in T.\ \varphi\ t \longrightarrow I_G\ t$, for all $B \in \mathcal{P}_{\mathsf{fin}}(Var)$ and $t \in T$, if φ is equivariant and $G\ \varphi\ B\ t$ then there exists $B' \in \mathcal{P}_{\mathsf{fin}}(Var)$ such that $B' \cap Supp^{\mathcal{T}}\ t = \emptyset$ and $G\ \varphi\ B'\ t$.

Since in the statement of $\mathcal{T}$-refreshability, φ morally stands for the inductively defined predicate I_G, adding the hypothesis that φ actually implies I_G makes sense. And indeed, with a bit of proof mining we can strengthen Thm. 7 to use this weaker notion:

Thm 7 strengthened. Let $\mathcal{T} = (T, \_[]^{\mathcal{T}})$ be a nominal set and $G : (T \to Bool) \to (\mathcal{P}_{\mathsf{fin}}(Var) \to T \to Bool)$ be monotonic, $\mathcal{T}$-equivariant and weakly $\mathcal{T}$-refreshable. Then Thm. 7's conclusion holds.

Proof. The only place in the proof of Thm. 7 where we use $\mathcal{T}$-refreshability is when proving **(3)** $\forall t.\ I_G\ t \longrightarrow I'_G\ t$, at a time when we have already proved the converse **(1)** $\forall t.\ I'_G\ t \longrightarrow I_G\ t$, and have also proved that **(2)** I'_G is equivariant. As part of the inductive proof of (3), fixing B and t and assuming **(iii)** $G\ I'_G\ B\ t$, we applied $\mathcal{T}$-refreshability to (2) and (iii) to obtain B' such that $B' \cap Supp^{\mathcal{T}}\ t = \emptyset$ and $G\ \varphi\ B'\ t$. But we can instead apply weak $\mathcal{T}$-refreshability to (2), (iii) and (1) to the same effect. □

In conclusion, the strengthened version of Thm. 7 assumes weak $\mathcal{T}$-refreshability instead of $\mathcal{T}$-refreshability, which allows one to take advantage of inductive information when instantiating the theorem. And indeed, the System $F_{<:}$ typing example is now covered, in that Prop. 14 is a consequence of the strengthened Thm. 7: Going back to the discussion at the end of §8.2, there the extra hypothesis $\forall t \in T.\ \varphi\ t \longrightarrow I_G\ t$ means that (ii) implies $\Gamma, X <: T_1 \vdash S_2 <: T_2$, which fills the pointed gap.

9 STRONG RULE INDUCTION FOR INFINITARY STRUCTURES WITH BINDINGS

While our strong induction criterion discussed so far covers the vast majority of the cases of interest, it is restricted to *finitary* structures modeled as nominal sets. However, infinitary structures featuring bindings have also been studied, and they too are subjected to inductive definitions and proofs that must cope with these bindings. Examples include infinitary extensions of first-order logic (FOL) [Dickmann 1985; Keisler 1971; Marker 2016] (§9.1), a standard variant of Milner's Calculus of Communicating Systems (CCS) [Milner 1989] featuring infinitary choice (sum) and bindings of input variables, versions of Hennessy-Milner logic featuring infinitary conjunctions and bindings for recursion and/or quantification [Hennessy and Stirling 1985], and infinitary higher-order rewriting and proof theory [Joachimski 2001]. Considering such infinitary logics and systems will lead to an extension of our result that employs an an infinitary variation of nominal sets (§9.2). Finally, we will

$$\frac{f \in \Delta}{\Delta \vdash f}\text{(Hyp)} \qquad \frac{\forall f \in F.\ \Delta \vdash f}{\Delta \vdash \mathit{Conj}\ F}\text{(Conj-I)} \qquad \frac{\Delta \vdash \mathit{Conj}\ F \quad f \in F}{\Delta \vdash f}\text{(Conj-E)} \qquad \frac{\Delta, f \vdash \bot}{\Delta \vdash \mathit{Neg}\ f}\text{(Neg-I)}$$

$$\frac{\Delta \vdash \mathit{Neg}\ f \quad \Delta \vdash f}{\Delta \vdash \bot}\text{(Neg-E)} \qquad \frac{\Delta \vdash f}{\Delta \vdash \mathit{All}\ V\ f}\ \begin{matrix}\text{(All-I)}\\ [V \cap \bigcup(\mathit{Im}\ \mathit{FV}\ \Delta) = \emptyset]\end{matrix} \qquad \frac{\Delta \vdash \mathit{All}\ V\ f}{\Delta \vdash f[\![\rho]\!]}\ \begin{matrix}\text{(All-E)}\\ [\mathit{Core}\ \rho \subseteq V]\end{matrix}$$

Fig. 8. Natural deduction system for $\mathcal{L}_{\kappa_1,\kappa_2}$. Equality rules omitted—they are the same as for standard FOL.

also look at situations that violate equivariance (§9.3), and show that such situations can still benefit from strong induction with the price of being explicit about the involved binding structures (§9.4).

9.1 Example: infinitary first-order logic

Given two infinite cardinals κ_1 and κ_2, the (κ_1,κ_2)-infinitary FOL logic $\mathcal{L}_{\kappa_1,\kappa_2}$ [Dickmann 1985; Keisler 1971; Marker 2016] is an extension of FOL which allows conjunctions / disjunctions of sets of formulas of any cardinality $< \kappa_1$, and quantifications over sets of variables of any cardinality $< \kappa_2$. ($\mathcal{L}_{\aleph_1,\aleph_0}$ is the best known version due to its importance for categorical logic [Makkai and Paré 1989].)

We let *Var* be an infinite set of cardinality $\kappa = \max(\kappa_1, \kappa_2)$. The set $\mathit{Fmla} = \mathit{Fmla}_{\kappa_1,\kappa_2}$ of $\mathcal{L}_{\kappa_1,\kappa_2}$-*formulas*, ranged over by f, is given by the grammar $f \ ::= \ \mathit{Eq}\ x\ y \mid \mathit{Neg}\ f \mid \mathit{Conj}\ F \mid \mathit{All}\ V\ f$ where F ranges (recursively) over $\mathcal{P}_{<\kappa_1}(\mathit{Fmla})$ (i.e., over sets of formulas of cardinality $< \kappa_1$) and V over $\mathcal{P}_{<\kappa_2}(\mathit{Var})$. Thus, a formula is either an equality, or a negation, or a conjunction over a set of formulas F, or a (simultaneous) quantification over a set of variables V. Again, formulas are identified modulo alpha-equivalence, e.g., $\mathit{All}\ \{x,y\}\ (\mathit{Eq}\ x\ y)$ and $\mathit{All}\ \{x,y\}\ (\mathit{Eq}\ y\ x)$ are the same formula.

Fig. 8 shows a straightforward generalization to $\mathcal{L}_{\kappa_1,\kappa_2}$ of the standard natural deduction rules for FOL, where Δ ranges over sets of formulas of cardinality $< \kappa$ and ρ over functions in $\mathit{Var} \to \mathit{Var}$. $\bot$ denotes the “false” formula, defined as $\mathit{Neg}\ (\mathit{Conj}\ \emptyset)$. Recall that $\mathit{Core}\ \rho$ denotes the core (support) of ρ, i.e., the set $\{x \in \mathit{Var} \mid \rho\ x \neq x\}$. Moreover, $\mathit{FV}\ f$ denotes the set of free variables of f, and $f[\![\rho]\!]$ denotes the (capture-free) parallel substitution of all free variables x in f with their ρ-image $\rho\ x$. The rules are standard except for accounting for the universal quantification of an entire set of variables V. Thus, the introduction rule (All-I) assumes freshness of all the variables in V for the hypotheses in Δ, and the elimination rule (All-E) makes sure that only variables in V are being instantiated.

By analogy with the finitary situations, we can hope to infer the following strong rule induction principle, which allows “avoiding” the bound variables V:

Prop 16. Let $(P, \mathit{Psupp} : P \to \mathcal{P}_{<\kappa}(\mathit{Var}))$ be a parameter structure. Let $\varphi : P \to \mathcal{P}_{<\kappa}\ \mathit{Fmla} \to \mathit{Fmla} \to \mathit{Bool}$ and assume that:

- [cases different from (|All-I|) and (|All-E|) omitted, as they don't involve binders]
- (|All-I|): $\forall p, \Delta, f.$
  $V \cap \mathit{Psupp}\ p = \emptyset \ \wedge\ V \cap \bigcup(\mathit{Im}\ \mathit{FV}\ \Delta) = \emptyset \ \wedge\ \Delta \vdash f \ \wedge\ (\forall q.\ \varphi\ q\ \Delta\ f) \longrightarrow \varphi\ \ p\ \Delta\ (\mathit{All}\ V\ f)$
- (|All-E|): $\forall p, \Delta, f.$
  $V \cap \mathit{Psupp}\ p = \emptyset \ \wedge\ \mathit{Core}\ \rho \subseteq V \ \wedge\ \Delta \vdash (\mathit{All}\ V\ f) \ \wedge\ (\forall q.\ \varphi\ q\ \Delta\ (\mathit{All}\ V\ f)) \longrightarrow \varphi\ p\ \Delta\ f$

Then $\forall p, \Delta, f.\ \Delta \vdash f \longrightarrow \varphi\ p\ \Delta\ f$.

9.2 An infinitary generalization of the criterion

So how do we go about obtaining Prop. 16 from the inductive definition of deduction in Fig. 8? Everything seems to follow our usual pattern, except that *Fmla* is no longer a nominal set but only a “nominal-set-like” structure, where sets are not finite but bounded by a cardinal κ. We say that a set is κ-*small* if its cardinality is $< \kappa$; so $\mathcal{P}_{<\kappa}(X)$ is the set of its κ-small subsets of a set X. Let us call

κ-permutation a bijection $\sigma : Var \to Var$ whose core is κ-small, and let $Perm_\kappa$ denote the set of κ-permutations. For formulas, the κ-permutation operator is here an action $\_[\_] : Fmla \to Perm_\kappa \to Fmla$ of $Perm_\kappa$ on $Fmla$; and the set of free variables $FV\ \varphi$ is no longer finite but only κ-small.

We therefore seek structures generalizing nominal sets in order to reach the following goals:

- (G1) Infinitary syntaxes with static bindings and their permutation and free-variable operators, such as $(Fmla, \_[\_] : Fmla \to Perm_\kappa \to Perm_\kappa, FV : Fmla \to \mathcal{P}_{<\kappa}(Var))$ above, should form such structures—i.e., the support operator should give exactly the free variables.
- (G2) Our strong rule induction criterion should carry over to these structures.
- (G3) Ideally, these structures should be closed under relevant constructions (such as sums, products, container type extensions)—similarly to standard nominal sets.

However, the naive generalization of nominal sets to higher cardinalities κ, replacing "finite" with "κ-smallness", does not work. We sketch it below, in preparation for something that will actually work. Let us call *κ-pre-nominal set* any pair $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ where $\_[\_]^{\mathcal{A}} : A \to Perm_\kappa \to A$ is an action on A of the monoid $(Perm_\kappa, 1_{Var}, \circ)$. Given a κ-pre-nominal set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$, an $a \in A$ and a set $X \subseteq Var$, we define the notion of *X supports a* by adapting that from nominal sets: as $a[\sigma]^{\mathcal{A}} = a$ holding for all $\sigma \in Perm_\kappa$ such that $\forall a \in X.\ \sigma\ x = x$ (i.e., $X \subseteq Core\ \sigma$). Finally, we define a *κ-nominal set* to be a κ-pre-nominal set where every element has a κ-small supporting set. Now, the problem is that a fundamental property of nominal sets does not carry over to κ-nominal sets $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ thus defined: Given $a \in A$, the least supporting set of a, which for nominal sets gave us the support $Supp^{\mathcal{A}}\ a$, is no longer guaranteed to exist. Here is a counterexample, which works for any $\kappa > \aleph_0$:

Counterexample 17. Let Var^∞ be the set of streams of variables. Given $xs \in Var^\infty$ and $i \in \mathbb{N}$, we write xs_i for the i'th variable in the stream. We say that two streams and ys are equivalent, written $xs \equiv ys$, if they are equal almost everywhere, i.e., there exists $n \in \mathbb{N}$ such that $xs_i = ys_i$ for all $i \geq n$. We let E be $Var^\infty/_\equiv$, the set of $\equiv$-equivalence classes. Given $xs \in Var^\infty$, we let $xs/_\equiv \in E$ denote its equivalence class. Since the standard permutation action on streams given by stream-map (so that $(xs[\sigma])_i = \sigma\ xs_i$ for each i) preserves $\equiv$, we can lift it to an operator on equivalence classes. This gives the κ-nominal set $\mathcal{E} = (E, \_[\_]^{\mathcal{E}})$ with $\_[\_]^{\mathcal{E}} : E \to Perm \to E$ defined as $(xs/_\equiv)[\sigma]^{\mathcal{E}} = (xs[\sigma])/_\equiv$. Now let $xs \in Var^\infty$ be any nonrepetitive stream. Each of the sets $\{x_i \mid i \geq n\}$ supports $xs/_\equiv$, but their intersection $\bigcap_{i\in\mathbb{N}}\{x_i \mid i \geq n\}$, which is empty, does not. So there is no least supporting set for $xs/_\equiv$.

Thus, if we switch from finite-core to κ-small-core permutations, we can no longer define the support as the least supporting set. But with goal (G2) in mind, we can ask whether our Thm. 7 really needs these least supporting sets or it can work with *any* supporting sets subject to weaker requirements. We discover these requirements looking back at Thm. 7's proof—where we have underlined the invocations of properties of the support operator $Supp = Supp^{\mathcal{T}}$ for the considered nominal set $\mathcal{T} = (T, \_[\_]^{\mathcal{T}})$. Fortunately, the minimality of $Supp$ is not needed in any of these. Rather:

- the last invocation of "properties of $Supp$" refers to the fact that $Supp$ returns supporting sets;
- the other invocations only require the property of the support being semi-natural w.r.t. permutation, in that $Supp\ (t[\sigma]) \subseteq Im\ \sigma\ (Supp\ t)$ for all $t \in T$ and $\sigma \in Perm$.

Thus, in the proof, we can replace the support operator with any operator satisfying the above two properties, which we will still call "support" (and denote by $Supp$). These more flexible assumptions allow a graceful transition from finiteness to κ-smallness. Indeed, our proof of Thm. 7 is resilient to this generalization as well: It only uses that finiteness is closed under permutation images and finite unions, which is also true about κ-smallness. This achieves goal (G2).

Remark 18. On the cardinality synchronization between support and permutations: For lifting the proof of Thm. 7 from finiteness to κ-smallness, it is essential that, in our generalization, permutations are allowed to "keep up" in cardinality with the support, in that the permutations now have κ-small cores (rather than just finite cores), matching the κ-smallness of the support. Indeed, the permutation τ that we use in the proof to "refresh" the set B' for avoiding *Psupp p* and *Supp* $(t[\sigma])$ (fact (vi)) must have its core's cardinality equal to that of B', which in the generalized version can be anything $< \kappa$.

Concerning goal (G1), it is easy to see that for the syntax of $\mathcal{L}_{\kappa_1,\kappa_2}$ (and any infinitary syntax for that matter), the free-variable operator *FV* satisfies the above desirable properties, in that *FV t* is a supporting set for t and $FV\ (t[\sigma]) \subseteq Im\ \sigma\ (Supp\ t)$. We are therefore led to the following definition, in the context of a fixed infinite cardinal κ and a fixed set of variables *Var* such that $|Var| = \kappa$.

Def 19. A κ*-loosely-supported-nominal set* (κ*-LS-nominal set* for short) is a triple $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ where $\_[\_]^{\mathcal{A}} : A \to Perm_\kappa \to A$ and $Supp^{\mathcal{A}} : A \to \mathcal{P}_{<\kappa}(Var)$ are such that:
- $(A, \_[\_]^{\mathcal{A}})$ is a κ-pre-nominal set i.e., $\_[\_]^{\mathcal{A}}$ is an action of the monoid $(Perm_\kappa, \circ, 1_A)$ on A;
- $Supp^{\mathcal{A}}$ returns supporting sets, i.e., $(\forall x \in Supp^{\mathcal{A}}.\ \sigma\ x = x)$ implies $a[\sigma]^{\mathcal{A}}$ for all a and σ;
- $Supp^{\mathcal{A}}$ is *semi-natural*, i.e., $Supp^{\mathcal{A}}\ (a[\sigma]) \subseteq Im\ \sigma\ (Supp^{\mathcal{A}}\ a)$ for all a and σ.

The "loosely" qualifier refers to the support operator $Supp^{\mathcal{A}}$ no longer being "tied" to give a specific supporting set (the least one). Note that, thanks to the κ-pre-nominal set axioms, semi-naturality is actually equivalent to naturality: $Supp^{\mathcal{A}}\ (a[\sigma]) = Im\ \sigma\ (Supp^{\mathcal{A}}\ a)$ for all a and σ.

So Thm. 7 generalizes to κ-LS-nominal sets. We work with κ-LS-nominal sets $\mathcal{T} = (T, \_[\_]^{\mathcal{T}}, Supp^{\mathcal{T}})$ instead of nominal sets $\mathcal{T} = (T, \_[\_]^{\mathcal{T}})$, and the bound-variable argument B of the operator G is now in $\mathcal{P}_{<\kappa}(Var)$ rather than $\mathcal{P}_{\mathsf{fin}}(Var)$. All the relevant notions, including equivariance and $\mathcal{T}$-refreshability, are defined like for nominal sets but replacing finiteness with κ-smallness.

Thm 20. Thm. 7 (also in its §8.3 strengthened form) still holds true if in its statement we replace:
- the nominal set $\mathcal{T} = (T, \_[\_]^{\mathcal{T}})$ and its support $Supp^{\mathcal{T}}$ with a κ-LS-nominal set $\mathcal{T} = (T, \_[\_]^{\mathcal{T}}, Supp^{\mathcal{T}})$;
- $G : (T \to Bool) \to (\mathcal{P}_{\mathsf{fin}}(Var) \to T \to Bool)$ with $G : (T \to Bool) \to (\mathcal{P}_{<\kappa}(Var) \to T \to Bool)$;
- the parameter structure $(P, Psupp : P \to \mathcal{P}_{\mathsf{fin}}\ (Var))$ with $(P, Psupp : P \to \mathcal{P}_{<\kappa}(Var))$.

So Thm. 20 (re)becomes Thm. 7 when $\kappa = \aleph_0$, and the κ-LS-nominal set $\mathcal{T} = (T, \_[\_]^{\mathcal{T}}, Supp^{\mathcal{T}})$ is a nominal set $\mathcal{T} = (T, \_[\_]^{\mathcal{T}})$ with its defined support operator. Moreover, when instantiating Thm. 20's operator G to that underlying the deduction system of $\mathcal{L}_{\kappa_1,\kappa_2}$, we obtain Prop. 16, as desired. Verifying the necessary hypotheses proceeds similarly to the finitary cases, via the §6 heuristic.

We have not yet addressed (G3), which bears upon the criterion's smooth instantiation, as it would allow constructing the required LS-nominal sets compositionally. It turns out that LS-nominal sets enjoy many of the closure properties of nominal sets [Pitts 2006; Urban 2008]. They are closed under the usual covariant set-theoretic (type-theoretic) constructions such as sums, products, and lifting via container types: both finitary ones such as lists, finite sets and bags, and infinitary ones such as streams, infinite trees, etc. (App. C gives details.)

In conclusion, we have extended our strong rule induction criterion to handle rule-based systems over infinitary structures with bindings, employing a mild extension of the nominal set axiomatization that still caters for concepts such as equivariance and refreshability. This should cover most of the infinitary situations of interest (including the ones cited at the beginning of §9). Our final stop in this paper is a case study where equivariance itself fails.

9.3 Example: an infinitary affine λ-calculus

In this subsection, *Var* will have cardinality $\aleph_1$, the first uncountable cardinal (so $\kappa = \aleph_1$). Recall that, A^∞ denotes the set of streams of elements in a set A, i.e. functions from $\mathbb{N}$ to A; we also let $A^{\infty,\neq}$ denote the subset of A^∞ consisting of the nonrepetitive streams, i.e., injective functions. Given $as \in A^\infty$, we write as_i for the i'th item in the stream, and *set as* for the set of its elements $\{as_i \mid i \in \mathbb{N}\}$ (its image as a function). We let *xs*, *ys* etc. range over the set $Var^{\infty,\neq}$ of nonrepetitive streams of variables.

$$affine\,(iVr\ x)\ \text{(iVr)} \qquad \frac{affine\ t}{affine\,(iLm\ xs\ t)}\ \text{(iLm)} \qquad \frac{affine\ t \quad lift\,(\lambda t'.\,affine\ t' \wedge FV\ t' \cap FV\ t = \emptyset)\ ts \quad \forall i, j.\ i \neq j \longrightarrow FV\ ts_i \cap FV\ ts_j = \emptyset}{affine\,(iAp\ t\ ts)}\ \text{(iAp)}$$

Fig. 9. The *affine* predicate

Following Mazza [2012], we define the syntax of *infinitary λ-calculus* by the following grammar, where t ranges over infinitary λ-terms (*λ-iterms*), i.e., elements of the syntax that is being introduced, and *ts* over streams of λ-iterms: $t\ ::=\ iVr\ x\ \mid\ iAp\ t\ ts\ \mid\ iLm\ xs\ t$. We assume that, in $iLm\ xs\ t$, the variables from the stream xs are bound in t; and λ-iterms are equated modulo the induced notion of alpha-equivalence. *ILTerm* denotes the set of λ-iterms. Given $t \in ILTerm$, $xs \in Var^{\infty,\neq}$ and $ts \in ILTerm^{\infty}$, we write $t[ts/xs]$ for the λ-iterm obtained by the simultaneous (capture-avoiding) substitution of the free occurrences in t of the variables xs_i with the corresponding λ-iterms ts_i.

Central in Mazza's development is the notion of a λ-iterm being *affine*, i.e., having no repeated occurrences of any free variable in it, or in any of its subterms (including subterms located under binders). This is expressed by the inductive predicate $affine : ILTerm \to Bool$ from Fig. 9, to which our Thm. 20 instantiates seamlessly, yielding the following strong induction principle. (Since $\kappa = \aleph_1$, $\mathcal{P}_{<\kappa}(Var)$ is $\mathcal{P}_{\text{countable}}(Var)$, the set of countable subsets of *Var*.)

Prop 21. Let $(P, Psupp : P \to \mathcal{P}_{\text{countable}}(Var))$ and $\varphi : P \to ILTerm \to Bool$, and assume that:
- [cases different from (|iLm|) omitted, as they don't involve binders]
- (|iLm|): $\forall p, xs, t.\ set\ xs\ \cap\ Psupp\ p = \emptyset \wedge affine\ t \wedge (\forall q.\ \varphi\ q\ t) \longrightarrow\ \varphi\ p\ (iLm\ xs\ t)$

Then $\forall p, t.\ affine\ t \longrightarrow \varphi\ p\ t$.

Since in our criterion the rules' hypotheses are not required to fit any syntactic format, higher-order operators and quantifiers such as Fig. 9's *lift* (which lifts a predicate from elements to streams, i.e., is defined by $lift\ \varphi\ as = (\forall i \in \mathbb{N}.\ \varphi\ as_i)$) can be used freely.

Mazza [2012]'s goal is to establish an isomorphic translation between (finitary) λ-calculus and a suitably *uniform* version of affine infinitary λ-calculus. This maps an application λ-term $Ap\ s\ t$ to an application λ-iterm $iAp\ s'\ ts'$, where s' is (recursively) an infinitary counterpart of s and ts' is a stream of copies of infinitary counterparts of t, with the copies having disjoint variables but otherwise having the same structure; and maps an abstraction λ-term $Lm\ x\ t$ to an abstraction λ-iterm $Lm\ xs'\ t'$, where t' is an infinitary counterpart of t and xs' is a nonrepetitive stream of copies of x.

To describe the image of this translation, Mazza fixes a countable subset $Super \subseteq Var^{\infty,\neq}$ of nonrepetitive streams of variables called *supervariables*, having the property that any two are mutually disjoint: $\forall xs, ys \in Super.\ set\ xs\ \cap\ set\ ys = \emptyset$. The intention is restricting the λ-iterms to only use these as bindings. Namely, supervariables induce the notion of *renaming equivalence* expressed as the relation $\approx\ : ILTerm \to ILTerm \to Bool$ which relates two λ-iterms t and t' just in case they **(1)** have the same (iVr, iLm, iAp)-structure (as trees), **(2)** only use supervariables in binders, **(3)** at the leaves have variables appearing in the same supervariable, and **(4)** for both t and t' all the subterms that form the righthand side of an application are mutually renaming equivalent. The $\approx$ relation is defined inductively in Fig. 10, via rules having a logical relation flavor. (Then *uniformity* of an λ-iterm, which together with affineness characterizes the translation's image, is defined as that λ-iterm being renaming-equivalent to itself. App. E gives details.)

Note that the set *Super* is not guaranteed to be closed under permutation. Even worse, it actually cannot be chosen so that it is closed, due to the disjointness assumption: If we permute some variables in a supervariable *xs* we obtain a stream of variables that is distinct but not disjoint from *xs*, which therefore cannot be a supervariable. For this reason, the monotonic operator underlying

$$\frac{xs \in \mathit{Super} \quad \{x, x'\} \subseteq \mathit{set}\ xs}{\mathit{iVr}\ x \ \approx\ \mathit{iVr}\ x'}\ (\mathrm{iVr}) \qquad \frac{xs \in \mathit{Super} \qquad t \approx t'}{\mathit{iLm}\ xs\ t \ \approx\ \mathit{iLm}\ xs\ t'}\ (\mathrm{iLm}) \qquad \frac{\begin{array}{c} t \approx t' \\ \forall t_1, t_2.\ \{t_1, t_2\} \subseteq \mathit{set}\ ts \cup \mathit{set}\ ts' \\ \longrightarrow t_1 \approx t_2 \end{array}}{\mathit{iAp}\ t\ ts \ \approx\ \mathit{iAp}\ t'\ ts'}\ (\mathrm{iAp})$$

Fig. 10. Mazza's renaming equivalence relation

$$G\ \varphi\ b\ (s, s') \iff \begin{array}{l} (1)\ \ (\exists xs, x, x'.\ b = \bot \wedge s = \mathit{iVr}\ x \wedge s' = \mathit{iVr}\ x' \wedge xs \in \mathit{Super} \wedge \{x, x'\} \subseteq \mathit{set}\ xs)\ \vee \\ (2)\ \ (\exists xs, t, t'.\ \varphi\ (t, t')\ \wedge b = xs \wedge s = \mathit{iLm}\ xs\ t \wedge s' = \mathit{iLm}\ xs\ t' \wedge xs \in \mathit{Super})\ \vee \\ (3)\ \ (\exists t, ts, t', ts'.\ \varphi\ (t, t') \wedge (\forall t_1, t_2.\ \{t_1, t_2\} \subseteq \mathit{set}\ ts \cup \mathit{set}\ ts' \\ \qquad\qquad \longrightarrow \varphi\ (t_1, t_2))\ \wedge b = \bot \wedge s = \mathit{iAp}\ t\ ts \wedge s' = \mathit{iAp}\ t'\ ts') \end{array}$$

Fig. 11. The operator associated to renaming equivalence

the definition of $\approx$, and the relation $\approx$ itself, are hopelessly non-equivariant, which renders strong induction impossible via current nominal criteria, including our own LS-nominal one. However, intuition tells us that when inducting over $\approx$ we should still be able to avoid the bound variables xs, similarly to how we did for *affine*, provided the parameters do not stretch too wide w.r.t. supervariables. And a reasonable notion of not stretching too wide is *touching only finitely many supervariables*. Thus, we can hope for the following strong induction principle for $\approx$, where the first highlighted part formalizes this condition regarding supervariables:

Prop 22. Let $(P, \mathit{Psupp} : P \longrightarrow \mathcal{P}_{\mathrm{countable}}(\mathit{Var}))$ be such that, for any $p \in P$, $\{xs \in \mathit{Super} \mid \mathit{set}\ xs \cap \mathit{Psupp}\ p \neq \emptyset\}$ is finite. Let $\varphi : P \longrightarrow \mathit{ILTerm} \longrightarrow \mathit{ILTerm} \longrightarrow \mathit{Bool}$ and assume the following:

- [cases different from (|iLm|) omitted, as they don't involve binders]
- (|iLm|): $\forall p, xs, t, t'.\ \mathit{set}\ xs \cap \mathit{Psupp}\ p = \emptyset \wedge xs \in \mathit{Super} \wedge t \approx t' \wedge (\forall q.\ \varphi\ q\ t\ t') \longrightarrow \varphi\ p\ (\mathit{iLm}\ xs\ t)\ (\mathit{iLm}\ xs\ t')$

Then $\forall p, t, t'.\ t \approx t' \longrightarrow \varphi\ p\ t\ t'$.

9.4 A criterion with explicit binders

The more general question we are led to is: *Can we still obtain strong induction in situations where equivariance fails, namely in the presence of non-equivariant restrictions on binders (such as the above supervariable restriction)?* To answer this, our Thm. 20's (and Thm. 7's) blurred view of binders needs to be sharpened. Indeed, the theorem refers to an inductive predicate's underlying operator G that acts not on binders directly, but on sets B of variables that are typically obtained by collecting the variables bound in the rules' conclusions; e.g., for the (iLm) rule for *affine* in Fig. 9, B is $\mathit{set}\ xs$. However, the set of variables in a binder can be oblivious to restrictions on binders, as is the case with supervariables in the $\approx$ example: two streams, one in and one not in *Super*, can have the same set of variables. Thus, when dealing with non-equivariant restrictions on binders, we must consider binders as first-class citizens. And LS-nominal sets again come handy for modeling this.

In addition to the κ-LS-nominal set of term-like items $\mathcal{T} = (T, \_[\_]^{\mathcal{T}}, \mathit{Supp}^{\mathcal{T}})$ (as before), we consider another κ-LS-nominal set $\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, \mathit{Supp}^{\mathcal{B}})$ of items that we will call "binders", and an operator $G : (T \to \mathit{Bool}) \to (B \to T \to \mathit{Bool})$. Provided G is monotonic, we again iterate it to define the predicate $I_G : T \longrightarrow \mathit{Bool}$ inductively by the rule $\frac{G\ I_G\ b\ t}{I_G\ t}$. To tackle the problem with non-equivariance, the key is to identify a suitable notion of *relative* equivariance, subject to sanity conditions w.r.t. freshness. We fix a predicate $\mathit{bnd} : B \longrightarrow \mathit{Bool}$ that singles out certain binders that are well-formed w.r.t. our considered inductive definition, and define $\mathit{Perm}_{\kappa,\mathit{bnd}}$ to be the set of κ-permutations $\sigma : \mathit{Var} \to \mathit{Var}$ that, applied via $\_[\_]^{\mathcal{B}}$, preserve well-formedness of binders, in that $\forall b \in B.\ \mathit{bnd}\ b \longrightarrow \mathit{bnd}\ (b[\sigma]^{\mathcal{B}})$. And we define *bnd-equivariance* by restricting equivariance to the bijections in $Perm_{\kappa,bnd}$. For example, a predicate $\varphi : T \to Bool$ is bnd-equivariant when $\varphi\ t$ implies $\varphi\ (t[\sigma]^{\mathcal{T}})$ for all $t \in T$ and $\sigma \in Perm_{\kappa,bnd}$.

We correspondingly generalize weak $\mathcal{T}$-refreshability: G is called is *weakly* $(\mathcal{T},\mathcal{B}, bnd)$*-refreshable* when, for all $\varphi : T \to Bool$, $b \in B$ and $t \in T$, if $\forall t \in T.\, \varphi\ t \longrightarrow I_G\, t$, φ is bnd-equivariant and $G\ \varphi\ b\ t$, then there exists $b' \in B$ with $Supp^{\mathcal{B}}\ b \cap Supp^{\mathcal{T}} t = \emptyset$ and $G\ \varphi\ b'\ t$.

Moreover, G is said to be *bnd-compatible* if it only holds for items satisfying the bnd restriction: $G\ R\ b\ t$ implies $bnd\ b$ for all R, b, t.

Finally, we want to be able express notions of size for our explicit binders that go beyond mere cardinality, such as "touching only finitely many supervariables". Rather than attempting to get too specific here, we employ an abstract predicate $bsmall : \mathcal{P}(Var) \to Bool$ (read "binder-small") subject to some sanity assumptions: $bsmall$ is said to be *closed under union* if $bsmall\ X$ and $bsmall\ Y$ implies $bsmall\ (X \cup Y)$ for all $X, Y \subseteq Var$. Moreover, I_G and bnd are said to be *bsmall-compatible* if $I_G\ t$ implies $bsmall\ (Supp^{\mathcal{T}}\, t)$ for all $t \in T$, and $bnd\ b$ implies $bsmall\ (Supp^{\mathcal{B}}\ b)$ for all $b \in B$, respectively.

The above generalizes our previous setting for strong rule induction, which can be obtained by taking $\mathcal{B}$ to be the κ-LS-nominal set having $B = \mathcal{P}_{<\kappa}(Var)$, and $\_[\_]^{\mathcal{B}}$ and $Supp^{\mathcal{B}}$ as the image and identity operators, respectively; and taking bnd and $bsmall$ to be vacuously true.

All the above assumptions should be expected to hold for most reasonable choices of the $bsmall$ predicate, and indeed they hold for the $\approx$ example if we take $bsmall$ to mean "touches only finitely many supervariables". So we can hope for a strong induction theorem that works when further restricting the parameters with a $bsmall$-ness assumption. And indeed, the proof of Thm. 20 (which was in turn adapted from that of Thm. 7) almost works, save for the step where we proved the existence of a permutation τ such that the facts labelled (vi) and (vii) hold. We want something similar to the cardinality reasoning invoked there, which applies to κ-smallness, to also apply to binder-smallness. We call the predicate bnd *bsmall-liftable* when the following condition holds: For all $A, A' \in \mathcal{P}_{<\kappa}(Var)$ and $b \in B$ such that $bsmall\ A$ and $bsmall\ A'$, if $A' \subseteq A$ and $Supp^{\mathcal{B}}\ b \cap A' = \emptyset$, then there exists $\tau \in Perm_{\kappa,bnd}$ such that $Im\ \tau\ (Supp^{\mathcal{B}}\ b) \cap A = \emptyset$ and $\forall x \in A'.\ \tau\ x = x$.

With these ingredients, we can prove a binder-explicit strong rule induction criterion. A parameter structure $\mathcal{P} = (P, Psupp : P \to \mathcal{P}_{<\kappa}(Var))$ is called *bsmall-compatible* if $bsmall\ (Psupp\ p)$ for all p.

Thm 23. Let $\mathcal{T} = (T, \_[\_]^{\mathcal{T}}, Supp^{\mathcal{T}})$ and $\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, Supp^{\mathcal{B}})$ be κ-LS-nominal sets, $bnd : B \to Bool$ and $bsmall : \mathcal{P}(Var) \to Bool$ predicates, and $G : (T \to Bool) \to (B \to T \to Bool)$ an operator, such that: **(1)** G is monotonic, bnd-compatible, bnd-equivariant and $(\mathcal{T},\mathcal{B}, bnd)$-refreshable; **(2)** $bsmall$ is closed under union; **(3)** I_G and bnd are $bsmall$-compatible; **(4)** bnd is $bsmall$-liftable.

Let $(P, Psupp)$ be a $bsmall$-compatible parameter structure and $\varphi : P \to T \to Bool$ such that:

$$\forall p \in P, t \in T, b \in B. \begin{pmatrix} Supp^{\mathcal{B}}\ b \cap (Psupp\ p \cup Supp^{\mathcal{T}} t) = \emptyset\ \wedge \\ G\ (\lambda t'.\ I_G\ t' \wedge \forall p' \in P.\ \varphi\ p'\ t')\ b\ t \end{pmatrix} \longrightarrow \varphi\ p\ t.$$

Then $\forall p \in P, t \in T.\ I_G\ t \longrightarrow \varphi\ p\ t.$

Thm. 23 is, by design, a generalization of Thm. 20. Also, it can be instantiated to obtain the desired strong induction for renaming equivalence, namely Prop. 21 (taking $\kappa = \aleph_1$):

- $\mathcal{T} = (T, \_[\_]^{\mathcal{T}}, Supp^{\mathcal{T}})$ taken as the $\aleph_1$-LS-nominal set structure on $T = ILTerm^2$;
- $\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, Supp^{\mathcal{B}})$ defined by taking $B = Var_{\bot}^{\infty,\neq} = Var^{\infty,\neq} \cup \{\bot\}$, where the elements of $Var^{\infty,\neq}$ are the proper binders and $\bot$ means "no binder"; and taking $\_[\_]^{\mathcal{B}}$ and $Supp^{\mathcal{B}}$ as the liftings to $Var_{\bot}^{\infty,\neq}$ of the map and set operators from $Var^{\infty,\neq}$;
- bnd defined to hold for $\bot$ and for any $xs \in Super$;
- $bsmall\ A$ defined as "$\{xs \in Super \mid set\ xs \cap A \neq \emptyset\}$ finite";
- G as shown in Fig. 11, making I_G (the uncurried version of) $\approx$.

The verification of Thm. 23's $\mathcal{T}$-refreshability assumption goes by a straightforward variation of our previous heuristic, working with permutations applied directly to binders (via $\_[\_]^{\mathcal{B}}$) rather than to sets of bound variables (via *Im*). Moreover, the *bnd*-compatibility of G, the closedness of *bsmall* under union, and the *bsmall*-compatibility of *bnd* are immediate; and the *bsmall*-compatibility of I_G follows by routine standard induction on I_G. The only non-routine check is that of the *bsmall*-liftability of *bnd*, which amounts to the following property: For all $xs \in \mathit{Super}$ and countable sets of variables A, A' that touch only finitely many supervariables and such that $A' \subseteq A$ and A' does not touch xs, there exists a supervariable-preserving permutation σ on variables such that $\{\sigma\ x \mid x \in \mathit{set}\ xs\} \cap A = \emptyset$ and $\mathit{Core}\,\sigma \cap A' = \emptyset$. This is proved by choosing a supervariable ys that is distinct (hence disjoint) from xs and is not touched by A, and defining τ to swap the elements of xs and ys componentwise and to be identity everywhere else (hence on A' too).

10 TOOL SUPPORT AND CASE STUDIES IN ISABELLE/HOL

We mechanized this paper's general theorems, instances and (counter) examples in Isabelle/HOL [Nipkow et al. 2002]. We further validated our induction principles in two proof developments: transitivity of the System $F_{<:}$ subtyping (part of POPLmark [Aydemir et al. 2005]) and the isomorphism between the affine uniform infinitary λ-calculus and the standard λ-calculus [Mazza 2012]. (Apps. §E, F and G.3 give details.) We also pursued an abstract case study: proving the rule-format based criterion of Urban et al. [2007] as an instance of our theorem. (App. A gives details.)

To support the use of the general theorems in concrete instances, we implemented a definitional extension of Isabelle's inductive specification and proof facilities, exported to users as new commands binder_datatype, binder_inductive, and make_binder_inductive, and the proof method binder_induction. The implementation and mechanization are available [van Brügge et al. 2025].

From a user specification of the syntax and its binders, the command binder_datatype defines the type of terms for that syntax quotiented to alpha-equivalence along the foundations sketched by Blanchette et al. [2019]. It also defines the constructors, renaming and free variable operators, proves their basic properties, and infers structural induction and recursion principles. We deployed it to obtain all this paper's datatypes: λ-terms, π-calculus processes and commitments, System $F_{<:}$ types, $\mathcal{L}_{\kappa_1,\kappa_2}$-formulas, and λ-iterms. (Apps. G.1 and D give details.)

Our general rule induction criteria, Thms. 7, 20 and 23, were formalized using Isabelle's locales [Ballarin 2014; Kammüller et al. 1999], a module system allowing to fix parameters, make assumptions about them, and infer consequences from these assumptions. For example, with Thm. 20 the parameters are the tuple $\mathcal{T} = (T, \_[\_]^{\mathcal{T}}, \mathit{Supp}^{\mathcal{T}})$ and the operator $G : (T \to \mathit{Bool}) \to (\mathcal{P}_{<\kappa}(\mathit{Var}) \to T \to \mathit{Bool})$, the assumptions are that $\mathcal{T}$ is a κ-LS-nominal set and G is monotonic, equivariant and (weakly) $\mathcal{T}$-refreshable; and the culmination of what is being inferred in that locale is the conclusion of Thm. 20, i.e., that the indicated strong rule induction holds for the predicate I_G defined inductively from G. Similarly for Thm. 7 and Thm. 23. Since Thm. 23 is more general than Thm. 20 which in turn is more general than Thm. 7, we only proved Thm. 23 directly and inferred Thm. 20 by showing how the former's parameters and assumptions can be instantiated to the latter's parameters and assumptions via a *sublocale* relationship (and similarly for inferring Thm. 7 from Thm. 20). Results stated in a locale can be obtained by *interpretation*, Isabelle's mechanism for instantiating a locale's parameters with concrete values and discharging the assumptions.

The commands binder_inductive and make_binder_inductive provide a high-level language for the user to endow an inductive predicates with a strong (binding-aware) rule induction principle (as an instance of our general result). binder_inductive behaves like the Isabelle/HOL inductive command for specifying standard inductive predicates by instantiating the Knaster-Tarski theorem (a command available in most HOL-based provers), but it additionally attempts to formulate and prove a strong rule induction principle. Namely, from a user specification of such a predicate using syntax identical to that required by the inductive command, our tool derives the relevant nominal set (or κ-LS-nominal set) infrastructure and the low-level operator G (as shown in this paper's examples), proves an instance of Thm. 20 for G, and outputs the strong induction theorem and other useful results such as the inductive predicate's equivariance. Currently, the tool automates the proofs of the (κ-LS-)nominal set axioms and equivariance, but requires the user to prove $\mathcal{T}$-refreshability—typically following the heuristic described in Section 6, which we have supported via some Isabelle tactics. The command make_binder_inductive is an incremental alternative to binder_inductive, allowing to decouple the standard inductive definition of a predicate (via "inductive") from its registration to produce a strong rule induction principle for it. Thus, issuing binder_inductive is equivalent to issuing an "inductive" followed by make_binder_inductive. The advantage of this decoupled approach is that in between the "inductive" and make_binder_inductive commands one can state and prove any inductive properties of the predicate needed in the proof of the assumptions for strong rule induction (as illustrated at the end of §8.2). The concrete strong rule induction priniciples for most examples (Props. 2, 13, 14, 16, 21, and all others mentioned in Apps. B and F) were obtained using `binder_inductive`. The strong rule induction principles requiring explicit binders (Thm. 23) such as Prop. 22 and others from App. E were obtained by manual locale interpretation.

Finally, our proof method binder_induction makes strong induction convenient to deploy in proofs. It allows the users to start induction while indicating the parameters to be avoided, as opposed to building the parameter structure explicitly.

We conclude with an example of our toolbox for the working syntax-with-bindings formalizer in action: the declaration of the datatype of System $F_{<:}$ types, the subtyping relation, and an example proof outline (of weakening of subtyping) with essential elements particular to our tools highlighted, namely the binding information for the datatype's constructors—here, the fact that the Forall constructor (denoted by ∀ in §8.2) binds the first (variable) argument into the third argument, and the parameters to be avoided when applying strong rule induction to prove weakening.

```
binder_datatype 'tvar sftypeP = TVr 'tvar | Top | Fun ('var sftypeP) ('var sftypeP)
    | Forall (x::'tvar) ('tvar typ) (t::'tvar typ) binds x in t
type_synonym sftype = tvar sftypeP
inductive ty :: (tvar × sftype) list → tvar sftype → tvar sftype → bool (_ ⊢ _ <: _) where
  SA_Top:        wf Γ ⟹ closed_in S Γ ⟹ Γ ⊢ S <: Top
| SA_Refl_TVar:  wf Γ ⟹ closed_in (TyVar x) Γ ⟹ Γ ⊢ TyVar x <: TyVar x
| SA_Trans_TVar: (x, U) ∈ set Γ ⟹ Γ ⊢ U <: T ⟹ Γ ⊢ TyVar x <: T
| SA_Arrow:      Γ ⊢ T₁ <: S₁ ⟹ Γ ⊢ S₂ <: T₂ ⟹ Γ ⊢ Fun S₁ S₂ <: Fun T₁ T₂
| SA_All:        Γ ⊢ T₁ <: S₁ ⟹ Γ;(x,T₁) ⊢ S₂ <: T₂ ⟹ Γ ⊢ Forall x S₁ S₂ <: Forall x T₁ T₂
⋮ 2 immediate lemmas about typing (mentioned at the end of §8.2) proved by rule induction
make_binder_inductive ty
    ⋮ 30 lines proof of weak 𝒯-refreshability using the heuristic (§6)
lemma ty_weakening: ⟦Γ ⊢ S <: T; ⊢ wf (Γ; Δ)⟧ ⟹ Γ; Δ ⊢ S <: T
proof (binder_induction Γ S T avoiding: dom Δ rule: ty_strong_induct)
    ⋮ 12 lines routine proof using the strong induction principle's Barendregt convention
```

Note that our datatype 'var sftypeP for System $F_{<:}$ types is polymorphic in the type 'tvar of (type) variables—and this is the case with all our datatypes for this paper's examples. This is to achieve slightly higher generality. Namely, instead of working with a fixed set of variables of suitable cardinality (which in the finitary case is just $\aleph_0$), that set is kept as a parameter—and in Isabelle/HOL, taking advantage of polymorphism, this is a type variable 'tvar of type class that specifies the cardinality constraint. (The binder_datatype command automatically assigns 'tvar to have the suitable type class.) This allows more flexibility in case we want to nest the given datatype inside another datatype that perhaps requires larger sets of variables. But once the exact datatypes needed for a case study have been decided, to cut down the unnecessary polymorphism we instantiate the type variables with fixed types; here, we instantiate 'tvar with a fixed type tvar of suitable cardinality (here, countable), and sftype is introduced as an Isabelle type synonym for tvar sftypeP, i.e., for the instance of the polymorphic type 'tvar sftypeP with the fixed type tvar. The subsequent inductive and make_binder_inductive commands shown above use this monomorphic type.

11 FURTHER RELATED WORK

The proximal related work is Urban et al. [2007], which we have extensively discussed throughout this paper. It is the literature's most general account of rule induction obeying Barendregt's variable convention. Its syntactic format criterion generalizes previous others, which operate on particular syntaxes [Bengtson 2010; McKinna and Pollack 1999; Norrish 2006; Urban and Norrish 2005].

Complementary to our work on binding-aware rule induction is work on binding-aware datatypes. This includes general mechanisms for building alpha-quotiented datatypes for binding signatures [Blanchette et al. 2019; Pitts 2006; Urban and Kaliszyk 2012], and also Barendregt-convention observing (strong) structural induction and recursion [Blanchette et al. 2019; Norrish 2004; Pitts 2006]. Since structural induction can be regarded as a particular case of rule induction (for the monotonic operator that applies the datatype's constructors), our work can be seen as generalizing the strong induction components of those works—although the main difficulty there lies with the construction of the datatypes and the inference of the recursion principles, which are orthogonal to our contribution.

Our tool described in §10 provides support for both binding-aware rule induction and binding-aware datatypes in Isabelle. It is more expressive than Nominal Isabelle [Urban and Tasson 2005] (including the Nominal 2 variant [Urban and Kaliszyk 2012]) in both the allowed datatypes and inductive predicates—reflecting the higher generality and flexibility of our criterion compared to Urban et al. [2007]. But it is currently in a prototype stage, lacking Nominal Isabelle's high degree of automation which has been finetuned based on feedback from its many users over the years. We are contemplating a future integration of these two tools, combining the best of both worlds.

We are not the first to relax the finite support assumption of nominal sets—Pitts [2013, §2.10] summarizes existing approaches. On the way to his completeness theorem for nominal logic, Cheney [2006] generalizes the support operator by noticing that the finite subsets of atoms (in our terminology, variables) $\mathcal{P}_{\mathsf{fin}}(\mathit{Var})$ form an ideal of $\mathcal{P}(\mathit{Var})$ that contains all singleton sets $\{x\}$, and replacing $\mathcal{P}_{\mathsf{fin}}(\mathit{Var})$ with an arbitrary such ideal $\mathcal{I}$, thus introducing $\mathcal{I}$-nominal sets—defined as pre-nominal sets such that every element has a supporting set of atoms from $\mathcal{I}$. The role of $\mathcal{I}$-nominal sets is that nominal logic deduction becomes complete w.r.t. these looser, ideal-supported models. Since $\mathcal{P}_{<\kappa}(\mathit{Var})$ is also such an ideal, Cheney's $\mathcal{I}$ -nominal sets cover structures with infinite support. However, regardless of the ideal $\mathcal{I}$ (be it $\mathcal{P}_{\mathsf{fin}}(\mathit{Var})$, or $\mathcal{P}_{<\kappa}(\mathit{Var})$, etc.), Cheney still defines the notion of supporting set using swapping, which is equivalent to using *finite*-core permutations, whereas we allow larger permutations whose cores have cardinality $< \kappa$. While staying with finite-core permutations was suitable for Cheney's goal of proving completeness, as we discuss in Remark 18 strong induction coping with κ-small support requires κ-small-core permutations. Since $\mathcal{I}$-nominal sets are (semi-)natural w.r.t. finite-core permutations only, a variation of our Thm. 20 would apply to $\mathcal{I}$-nominal sets if we restricted the parameters to be finitely supported (i.e., *Psupp* to return finite sets). But being able to avoid only finitely many variables when proving facts about structures having infinitely many (free) variables would not be very useful.

Dowek and Gabbay [2012] introduce *permissive nominal sets*, a generalization of nominal sets based on separating atoms (variables) in two categories, along the distinction between free and bound variables. The elements in permissive nominal sets have supporting *permission sets*, which contain finitely many atoms of one category and co-finitely many of the other; this ensures the existence of least supporting sets. Like with Cheney's $\mathcal{I}$-nominal sets and differently from our κ-LS-nominal sets, the notion of supporting set is defined there using finite-core permutations. Permissive nominal sets are the semantic underpinning of *permissive nominal logic* [Dowek and Gabbay 2012, 2023; Dowek et al. 2010], an elaborate extension of nominal logic with enhanced support for contextual and higher-order reasoning. Another difference between both $\mathcal{I}$-nominal sets and permissive nominal sets and our LS-nominal sets is that, the former retain the minimality of the support whereas in the latter we replaced minimality with the weaker axiom of (semi-)naturality.

Gabbay [2007] develops a nominal-style axiomatic set theory, FMG (Fraenkel-Mostowski Generalized), which generalizes the Fraenkel-Mostowski set theory previously introduced by Gabbay and Pitts [2002] as a foundation for nominal logic. In FMG, "smallness" of a set (such as the support of an item) no longer means "finiteness", but the possibility to internally well-order that set. This covers in particular cardinality bounds like the ones we use in LS-nominal sets. When developing his theory, Gabbay also constructs datatypes and develops mechanisms for extending functions from representatives to equivalence classes (via his *Barendregt abstractive* functions). Our preliminary investigations suggest that our criterion for strong rule induction could be adapted to Gabbay's FMG, complementing his results about datatypes and recursive-function definition principles.

Like the above works, we operate within (a transfinite generalization of) the nominal paradigm, where the names of the variables are visible, but ultimately irrelevant in that their choice does not matter. Barendregt's convention only makes sense in this paradigm. The other two major paradigms on representing and reasoning about syntax with bindings are based on nameless / De Bruijn representations [de Bruijn 1972] (and its type-safe and scope-safe generalizations, e.g., [Allais et al. 2018; Fiore et al. 1999; Schäfer et al. 2015]) and higher-order abstract syntax (HOAS) [Baelde et al. 2014; Harper et al. 1987; Pfenning and Elliott 1988; Pfenning and Schürmann 1999; Pientka 2010]. (Cross-paradigm hybrids have also been proposed, e.g., [Aydemir et al. 2008; Charguéraud 2012; Felty and Momigliano 2012; McKinna and Pollack 1999; Pollack et al. 2012].) There are relative pros and cons between these paradigms [Abel et al. 2017; Berghofer and Urban 2006; Felty and Momigliano 2012; Gheri and Popescu 2020; Kaiser et al. 2017; Norrish and Vestergaard 2007]. An advantage of the nominal paradigm is faithfulness to the informal, textbook descriptions of the systems. Our contribution is also in this direction, by lowering the informal-formal gap in nominal-style strong rule induction.

While our LS-nominal sets accommodate both infinitely branching and infinitely deep (non-well-founded) syntax, our infinitary examples (in §9.1, §9.3 and App. E) only involve the former. The latter also has a rich literature—centered around concepts such as Böhm, Lévy-Longo and Berarducci trees [Barendregt and Klop 2009; Berarducci and Dezani-Ciancaglini 1999], used in the λ-calculus semantics. While inductively defined predicates on non-well-founded trees will fall under our strong induction criterion, such structures are often best explored not inductively, but coinductively, i.e., via predicates defined not as least but as greatest fixed points. We leave as future work the study of Barendregt's variable convention for rule-based coinduction. This would complement existing results on nominal-syle codatypes and corecursion [Blanchette et al. 2019; Kurz et al. 2012, 2013; Milius and Wißmann 2015; Popescu 2024].

Acknowledgments. We are grateful to the reviewers for their excellent questions and suggestions for improvement, which we were able to factor in thanks to the ample additional space available for the final version. Popescu acknowledges support from the EPSRC grant EP/X015114/1, titled "Safe and secure COncurrent programming for adVancEd aRchiTectures (COVERT)". Traytel acknowledges support from the Novo Nordisk Foundation (start package grant NNF20OC0063462) and the Independent Research Fund Denmark (DFF Sapere Aude grant 3120-00057B, titled DISCOVER). The authors are listed alphabetically regardless of individual contributions or seniority.

REFERENCES

2024. The HOL4 Theorem Prover. http://hol.sourceforge.net/.

Michael Gordon Abbott, Thorsten Altenkirch, and Neil Ghani. 2005. Containers: Constructing strictly positive types. *Theor. Comput. Sci.* 342, 1 (2005), 3–27. https://doi.org/10.1016/J.TCS.2005.06.002

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

Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. 2018. A Type and Scope Safe Universe of Syntaxes with Binding: Their Semantics and Proofs. *Proc. ACM Program. Lang.* 2, International Conference on Functional Programming (ICFP) (2018), 90:1–90:30. http://doi.acm.org/10.1145/3236785

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, 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

Henk Barendregt and Jan Willem Klop. 2009. Applications of infinitary lambda calculus. *Inf. Comput.* 207, 5 (2009), 559–582. https://doi.org/10.1016/J.IC.2008.09.003

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

Jesper Bengtson. 2010. *Formalising process calculi*. Ph. D. Dissertation. Uppsala University, Sweden. http://www.itu.dk/people/jebe/files/thesis.pdf

Jesper Bengtson. 2012. The pi-calculus in nominal logic. *Arch. Formal Proofs* 2012 (2012). https://www.isa-afp.org/entries/Pi_Calculus.shtml

Alessandro Berarducci and Mariangiola Dezani-Ciancaglini. 1999. Infinite lambda-Calculus and Types. *Theor. Comput. Sci.* 212, 1-2 (1999), 29–75. https://doi.org/10.1016/S0304-3975(98)00135-2

Stefan Berghofer and Christian Urban. 2006. A Head-to-Head Comparison of de Bruijn Indices and Names. In *LFMTP 2006 (ENTCS, Vol. 174)*, Alberto Momigliano and Brigitte Pientka (Eds.). Elsevier, 53–67. https://doi.org/10.1016/j.entcs.2007.01.018

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

Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. 1994. An Extension of System F with Subtyping. *Inf. Comput.* 109, 1/2 (1994), 4–56. https://doi.org/10.1006/inco.1994.1013

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

James Cheney. 2006. Completeness and Herbrand theorems for nominal logic. *J. Symb. Log.* 71, 1 (2006), 299–320. https://doi.org/10.2178/JSL/1140641176

Pierre-Louis Curien and Giorgio Ghelli. 1992. Coherence of Subsumption, Minimum Typing and Type-Checking in $F_{<=}$. *Math. Struct. Comput. Sci.* 2, 1 (1992), 55–91. https://doi.org/10.1017/S0960129500001134

N. G. de Bruijn. 1972. Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church–Rosser Theorem. *Indag. Math* 75, 5 (1972), 381–392. https://doi.org/10.1016/1385-7258(72)90034-0

M. Dickmann. 1985. Larger infinitary languages. In *Handbook of Model Theoretic Logics*. Springer-Verlag, 150–171.

Gilles Dowek and Murdoch James Gabbay. 2012. Permissive-nominal logic: First-order logic over nominal terms and sets. *ACM Trans. Comput. Log.* 13, 3 (2012), 20:1–20:36. https://doi.org/10.1145/2287718.2287720

Gilles Dowek and Murdoch James Gabbay. 2023. PNL to HOL: from the logic of nominal sets to the logic of higher-order functions. *CoRR* abs/2312.16239 (2023). https://doi.org/10.48550/ARXIV.2312.16239 arXiv:2312.16239

Gilles Dowek, Murdoch James Gabbay, and Dominic P. Mulligan. 2010. Permissive nominal terms and their unification: an infinite, co-infinite approach to nominal techniques. *Log. J. IGPL* 18, 6 (2010), 769–822. https://doi.org/10.1093/JIGPAL/JZQ006

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

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. 2007. A general mathematics of names. *Inf. Comput.* 205, 7 (2007), 982–1011. https://doi.org/10.1016/J.IC.2006.10.010

Murdoch Gabbay and Andrew M. Pitts. 1999. A New Approach to Abstract Syntax Involving Binders. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. IEEE Computer Society, 214–224. https://doi.org/10.1109/LICS.1999.782617

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

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

M. J. C. Gordon and T. F. Melham (Eds.). 1993. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press.

William P. Hanf. 1964. Languages with Expressions of Infinite Length. *Journal of Symbolic Logic* 33, 3 (1964), 477–478. https://doi.org/10.2307/2270356

Robert Harper, Furio Honsell, and Gordon D. Plotkin. 1987. A Framework for Defining Logics. In *Logic in Computer Science (LICS) 1987*. IEEE Computer Society, 194–204. https://doi.org/10.1145/138027.138060

John Harrison. 2024. The HOL Light Theorem Prover. http://www.cl.cam.ac.uk/~jrh13/hol-light/.

Matthew Hennessy and Colin Stirling. 1985. The Power of the Future Perfect in Program Logics. *Inf. Control.* 67, 1-3 (1985), 23–52. https://doi.org/10.1016/S0019-9958(85)80025-5

Brian Huffman and Ondřej Kunčar. 2013. Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In *Certified Programs and Proofs*, Georges Gonthier and Michael Norrish (Eds.). Springer International Publishing, Cham, 131–146. https://doi.org/10.1007/978-3-319-03545-1_9

Felix Joachimski. 2001. *Reduction Properties of ΠΙΕ-Systems*. Ph. D. Dissertation. LMU München.

Jonas Kaiser, Brigitte Pientka, and Gert Smolka. 2017. Relating System F and $\lambda 2$: A Case Study in Coq, Abella and Beluga. In *Formal Structures for Computation and Deduction (FSCD) 2017*, Dale Miller (Ed.). LIPIcs, Vol. 84. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 21:1–21:19. https://doi.org/10.4230/LIPIcs.FSCD.2017.21

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, Vol. 1690)*, Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin-Mohring, and Laurent Théry (Eds.). Springer, 149–166. https://doi.org/10.1007/3-540-48256-3_11

H. Jerome Keisler. 1971. *Model Theory for Infinitary Logic*. North-Holland Pub. Co., Amsterdam,.

B. Knaster. 1928. Un théorème sur les fonctions d'ensembles. *Ann. Soc. Polon. Math.* 6 (1928), 133–134.

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

Jean-Jacques Lévy. 1975. An algebraic interpretation of the lambda beta - calculus and a labeled lambda - calculus. In *Lambda-Calculus and Computer Science Theory, Proceedings of the Symposium Held in Rome, Italy, March 25-27, 1975 (Lecture Notes in Computer Science, Vol. 37)*, Corrado Böhm (Ed.). Springer, 147–165. https://doi.org/10.1007/BFb0029523

Michael Makkai. 1969. On the Model Theory of Denumerably Long Formulas with Finite Strings of Quantifiers. *J. Symb. Log.* 34, 3 (1969), 437–459. https://doi.org/10.2307/2270908

Michael Makkai and Robert Paré. 1989. *Accessible Categories: The Foundations of Categorical Model Theory*. Providence.

David Marker. 2016. *Lectures on Infinitary Model Theory*. Cambridge University Press, New York, NY, USA.

Damiano Mazza. 2012. An Infinitary Affine Lambda-Calculus Isomorphic to the Full Lambda-Calculus. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*. 471–480. https://doi.org/10.1109/LICS.2012.57

James McKinna and Robert Pollack. 1999. Some Lambda Calculus and Type Theory Formalized. *J. Autom. Reason.* 23, 3-4 (1999), 373–409.

Stefan Milius and Thorsten Wißmann. 2015. Finitary Corecursion for the Infinitary Lambda Calculus. In *6th Conference on Algebra and Coalgebra in Computer Science, CALCO 2015, June 24-26, 2015, Nijmegen, The Netherlands (LIPIcs, Vol. 35)*, Lawrence S. Moss and Pawel Sobocinski (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 336–351. https://doi.org/10.4230/LIPICS.CALCO.2015.336

R. Milner. 1989. *Communication and Concurrency*. Prentice-Hall, Inc., USA.

Robin Milner. 1993. The Polyadic π-Calculus: a Tutorial. In *Logic and Algebra of Specification*, Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 203–246.

Robin Milner. 1999. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press.

Robin Milner, Joachim Parrow, and David Walker. 1992. A Calculus of Mobile Processes, I/II. *Inf. Comput.* 100, 1 (1992), 1–77. https://doi.org/10.1016/0890-5401(92)90008-4

Tobias Nipkow, Lawrence C. Paulson, and Markarius Wenzel. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Vol. 2283. Springer Science & Business Media.

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. 2006. Mechanising lambda-calculus using a classical first order theory of terms with permutations. *High. Order Symb. Comput.* 19, 2-3 (2006), 169–195. https://doi.org/10.1007/S10990-006-8745-7

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, Vol. 4732)*, Klaus Schneider and Jens Brandt (Eds.). Springer, 207–222. https://doi.org/10.1007/978-3-540-74591-4_16

Frank Pfenning and Conal Elliott. 1988. Higher-Order Abstract Syntax. In *Programming Language Design and Implementation (PLDI) 1988*, Richard L. Wexelblat (Ed.). ACM, 199–208. https://doi.org/10.1145/53990.54010

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. 2003. Nominal Logic, a First Order Theory of Names and Binding. *Inf. Comput.* 186, 2 (2003), 165–193. https://doi.org/10.1016/S0890-5401(03)00138-X

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

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. 2024. Nominal Recursors as Epi-Recursors. *Proc. ACM Program. Lang.* 3, POPL (2024), 22:1–22:34. https://doi.org/10.1145/3290335

John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism.. In *IFIP Congress*. 513–523.

Davide Sangiorgi and David Walker. 2001. *The π-calculus. A theory of mobile processes.* Cambridge.

Steven Schäfer, Tobias Tebbi, and Gert Smolka. 2015. Autosubst: Reasoning with de Bruijn Terms and Parallel Substitutions. In *ITP 2015 (LNCS, Vol. 9236)*, Christian Urban and Xingyuan Zhang (Eds.). Springer, 359–374. https://doi.org/10.1007/978-3-319-22102-1_24

Masako Takahashi. 1995. Parallel Reductions in lambda-Calculus. *Inf. Comput.* 118, 1 (1995), 120–127. https://doi.org/10.1006/inco.1995.1057

Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.* 5 (1955), 285–309. https://api.semanticscholar.org/CorpusID:13651629

Dmitriy Traytel, Andrei Popescu, and Jasmin Christian Blanchette. 2012. Foundational, Compositional (Co)datatypes for Higher-Order Logic: Category Theory Applied to Theorem Proving. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. IEEE Computer Society, 596–605. https://doi.org/10.1109/LICS.2012.75

Christian Urban. 2008. Nominal Techniques in Isabelle/HOL. *J. Autom. Reason.* 40, 4 (2008), 327–356. https://doi.org/10.1007/S10817-008-9097-2

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 Michael Norrish. 2005. A formal treatment of the Barendregt variable convention in rule inductions. In *ACM SIGPLAN International Conference on Functional Programming, Workshop on Mechanized reasoning about languages with variable binding, MERLIN 2005, Tallinn, Estonia, September 30, 2005*, Randy Pollack (Ed.). ACM, 25–32. https://doi.org/10.1145/1088454.1088458

Christian Urban and Christine Tasson. 2005. Nominal Techniques in Isabelle/HOL. In *CADE*. 38–53.

Jan van Brügge, James McKinna, Andrei Popescu, and Dmitriy Traytel. 2025. Barendregt Convenes with Knaster and Tarski: Implementation and Mechanization Artifact. https://github.com/jvanbruegge/binder_datatypes/tree/popl2025_updated (will be replaced by a Zenodo link after artifact evaluation).

Philip Wadler. 1989. Theorems for Free!. In *FPCA '89*. ACM, 347–359.

APPENDIX

This appendix provides more details, extensions and case studies pertaining to the concepts and results presented in the main paper. Specifically, it provides:

- a formal account of Urban et al.'s strong rule induction criterion (§A) including a proof that it is subsumed by our criterion;
- the application of our criterion to an alternative variant of π-calculus that distinguishes between structural and operational rules §B
- more details on κ-LS-nominal sets, including a statement and proof sketch of their closure properties (§C);
- details on the datatypes of terms with bindings used in our examples, namely finitary and infinitary λ-calculus terms, and π-calculus processes (§D);
- a formal proof development, taking advantage of our strong rule induction infrastrucrure (as well as of some datatype-specific infrastructure), leading to the isomorphism between affine uniform infinitary λ-calculus and finitary λ-calculus established by Mazza [2012] (§E);
- a formal proof of the transitivity of the subtyping relation for System $F_{<:}$, a smaller case study (§F);
- a description of our Isabelle implementation and formalization of the case studies (§G).

A THE URBAN ET AL. PRINCIPLE

In this section we present a formalization of Urban et al. [2007]'s strong rule induction criterion and show that it follows as an instance of our criterion. For the whole section, we assume that *Var* is countable.

Urban et al. describe their criterion using schematic rules. To formalize these, we fix two infinitely countable sets:

- *VMVar*, ranged over by u, v, of *variable metavariables*
- *TMVar*, ranged over by U, V, of *term metavariables*

A signature is a pair $\Sigma = (Sym, arOf : Sym \to \mathbb{N})$ where *Sym*, ranged over by σ, is a set of items called *operation symbols* and *arOf* associates numeric arities to them. The *schematic terms (sterms) over* Σ, forming the set $SchTerm(\Sigma)$, ranged over by s, s' etc., are generated by the following grammar:

$$s \quad ::= \quad VVr\ u \mid TVr\ U \ \mid\ SAbs\ u\ s \ \mid\ SOp\ \sigma\ (s_1, \ldots, s_{arOf\ \sigma})$$

Thus, an sterm is either a variable metavariable, or a term metavariable, or recursively a schematic abstraction of a variable metavariable in an sterm, or recursively an operation symbol applied (symbolically) to a tuple of sterms of length matching the symbol's arity.

Example 24. Assume $\Sigma = \{ap, sub\}$, and assume $arOf\ ap = 2$ and $arOf\ sub = 3$. Given variable metavariables u and term metavariables U, U', we have that

$$SOp\ ap\ (SAbs\ u\ (TVr\ U), TVr\ U')$$

is an sterm, which can be thought of as a schematic representation of a λ-term of the form $Ap\ (Lm\ x\ t)\ t'$ for unspecified variable x and terms t, t'. Moreover,

$$SOp\ sub\ (TVr\ U')\ (TVr\ U)\ (VVr\ u)$$

is an sterm, which can represent a term of the form $t'[t/x]$, again for unspecified variable x and terms t, t'. (Such intuitive readings will be made precise below using enriched nominal sets and interpretations.) ◻

In their criterion, Urban et al. implicitly refer to interpretations of the schematic terms as concrete term-like entities (inhabitants of nominal sets, such as the set of λ-terms). In order to formalize their criterion and compare it to ours, we will need to make these interpretations explicit.

Def 25. Given a signature $\Sigma = (Sym, arOf)$, a Σ-*enriched nominal set* is a tuple $\mathcal{T} = (T, \_[\_]^{\mathcal{T}}, Vr, Abs, Op)$ where $(T, \_[\_]^{\mathcal{T}})$ is a nominal set, and $Vr : Var \to T$, $Abs : Var \to T \to T$ and $Op : \sum_{\sigma \in Sym}(T^{arOf\ \sigma} \to T)$ are some operators. We will write $\_[\_]$ instead of $\_[\_]^{\mathcal{T}}$.

Schematic terms are naturally interpreted in Σ-enriched nominal sets $\mathcal{T} = (T, \_[\_], Vr, Abs, Op)$, in the context of:

- valuations $\rho : VMVar \to Var$ of the variable metavariables as variables and
- valuations $\delta : TMVar \to T$ of the term metavariables as "term-like entities" provided by the nominal set, i.e., elements of T.

Namely, the interpretation function

$$int_{\mathcal{T}} : (VMVar \to Var) \to (TMVar \to T) \to SchTerm(\Sigma) \to T$$

has the expected recursive definition, where ρ is applied to variable metavariables leaves and abstractions, δ is applied to term metavariable leaves, and SVr, $SAbs$ and SOp are interpeted as Vr, Abs and Op:

- $int_{\mathcal{T}}\ \rho\ \delta\ (VVr\ u) = Vr\ (\rho\ u)$
- $int_{\mathcal{T}}\ \rho\ \delta\ (TVr\ U) = \delta\ U$
- $int_{\mathcal{T}}\ \rho\ \delta\ (SAbs\ u\ s) = Abs\ (\rho\ u)\ (int_{\mathcal{T}}\ \rho\ \delta\ s)$
- $int_{\mathcal{T}}\ \rho\ \delta\ (SOp\ \sigma\ (s_1, \ldots, s_{arOf\ \sigma})) = Op\ \sigma\ (int_{\mathcal{T}}\ \rho\ \delta\ s_1, \ldots, int_{\mathcal{T}}\ \rho\ \delta\ s_{arOf\ \sigma})$

The interpretation is extended from sterms to tuples of sterms componentwise:

$$int_{\mathcal{T}}\ \rho\ \delta\ (s_1, \ldots, s_k) = (int_{\mathcal{T}}\ \rho\ \delta\ s_1, \ldots, int_{\mathcal{T}}\ \rho\ \delta\ s_k)$$

Example 26. In the context of Example 24, taking $\mathcal{T}$ to be the nominal set of λ-terms, in particular taking $T = LTerm$, and taking:

- Vr to be the injection of variables into λ-terms (also denoted by Vr),
- $Abs\ x\ t$ to be $Lm\ x\ t$,
- $Op\ ap\ (t_1, t_2) = Ap\ t_1\ t_2$, and $Op\ sub\ (t_1, t_2, Vr\ x) = t_1[t_2/x]$.

and assuming $\rho\ u = x$, $\delta\ U = t$ and $\delta\ U' = t'$, then we have

$$int_{\mathcal{T}}(SOp\ ap\ (SAbs\ u\ (TVr\ U), TVr\ U')) = Ap\ (Lm\ x.\ t)\ t'$$

and $int_{\mathcal{T}}(SOp\ sub\ (TVr\ U')\ (TVr\ U)\ (VVr\ u)) = t'[t/t']$. □

Def 27. A *schematic rule* over Σ and n is a triple of the form $(hyps, conc, side)$ where

- $hyps = (hyps_1, \ldots, hyps_k)$, the *hypotheses*, is a sequence of n-tuples of sterms, $hyps_i = (s_{i,1}, \ldots, s_{i,n})$;
- $conc$, the *conclusion*, is an n-tuple of sterms, $conc = (s'_1, \ldots, s'_n)$;
- $side = (side_1, \ldots, side_l)$, the *side-condition*, is triple of pairs $side_i = (sideT_i, sideV_i, sideP_i)$ where:
  - $sideT_i \in SchTerm(\Sigma)^{r_i}$ is a tuple of schematic terms (say, if size r_i);
  - $sideV_i \in VMVar^{q_i}$ is a tuple of variable metavariables (say, if size q_i);
  - $sideP_i : T^{r_i} \to Var^{q_i} \to Bool$ is a predicate on tuples of terms and tuples of variables of arities matching the sizes of the above tuples;

$$\frac{\begin{array}{c}((hyps_1, \ldots, hyps_k), conc, (side_1, \ldots, side_l)) \in Rls \\ J_{Rls,\mathcal{T}}(int_{\mathcal{T}}\, \rho\, \delta\ hyps_1) \quad \ldots \quad J_{Rls,\mathcal{T}}(int_{\mathcal{T}}\, \rho\, \delta\ hyps_k)\end{array}}{J_{Rls,\mathcal{T}}(int_{\mathcal{T}}\, \rho\, \delta\ conc)} \left[\bigwedge_{i=1}^{l} sideP_i\ (int_{\mathcal{T}}\, \rho\, \delta\ sideT_i)\ (\overline{\rho}\ sideV_i)\right]$$

Fig. 12. The inductive predicate $J_{Rls,\mathcal{T}}$ induced by Rls over $\mathcal{T}$

A schematic rule $(hyps, conc, side)$ is meant to be visualized as follows:

$$\frac{hyps_1 \ldots hyps_k}{conc}\,[side]$$

We think of it as allowing one to infer the conclusion from the hypotheses in the presence of the side-conditions. This intuition is made precise below, where we define the inductive predicate induced by applying concrete interpretations of the schematic rules.

Given $\rho : VMVar \longrightarrow Var$ and any number q, we write $\overline{\rho}$ for the componentwise extension of ρ to $VMVar^q \longrightarrow Var^q$.

Def 28. Let Rls be a set of schematic rules over Σ and n, and let $\mathcal{T} = (T, \_[\_], Vr, Abs, Op)$ be a Σ-enriched nominal set. We define $J_{Rls,\mathcal{T}} : T^n \longrightarrow Bool$, the *inductive predicate induced by Rls over* $\mathcal{T}$, to be the least (pre)fixpoint obtained by applying the schematic rules interpreted in all possible ways, as shown in Fig. 12.

Example 29. We place ourselves in the context of Example 26. The rule (ParBeta') from §8, namely

$$\frac{t_1 \Rightarrow t_1' \qquad t_2 \Rightarrow t_2'}{Ap\ (Lm\ x\ t_1)\ t_2 \Rightarrow t_1'[t_2'/x]}\ \begin{array}{l}\text{(ParBeta')} \\ [x \notin FV\ t_2 \cup FV\ t_2']\end{array}$$

can be expressed as the schematic rule SParBeta' = $(hyps, conc, side)$ where:

- $hyps = (hyps_1, hyps_2)$ (so, in the notations of Def. 27, $k = 2$), where $hyps_1 = (TVr\ U_1, TVr\ U_1')$ and $hyps_2 = (TVr\ U_2, TVr\ U_2')$ for some fixed distinct term metavariables U_1, U_1', U_2, U_2';
- $conc = (s_1, s_2)$ where s_1 and s_2 are the following schematic terms:
  - $s_1 = SOp\ ap\,(SAbs\ u\ TVr\ U_1, TVr\ U_2)$;
  - $s_2 = SOp\ sub\,(TVr\ U_1', TVr\ U_2', SVr\ u))$ for a fixed variable metavariable u;
- $side = (side_1)$ (so $l = 1$) and $side_1 = (sideT_1, sideV_1, sideP_1)$, where $sideT_1 = (TVr\ U_2, TVr\ U_2')$ (a 2-ary tuple, so $r_1 = 2$), $sideV_1 = (u)$ (a singleton tuple, so $q_1 = 1$), and $sideP_1 : T^2 \longrightarrow Var \longrightarrow VarT$ is defined by $sideT_1\ (t, t')\ x = (x \notin FV\ t \cup FV\ t')$.

So if we write this schematic rule in the form

$$\frac{hyps_1 \ldots hyps_k}{conc}\,[side]$$

more precisely, in the form

$$\frac{hyps_1 \quad hyps_2}{conc}\ \begin{array}{l}[(sideT_1, \\ \ \ sideV_1, \\ \ \ sideP_1)]\end{array}$$

we get obtain what is shown in Fig. 13.

One can check that, when applying Def. 28 to the above schematic rule SParBeta' (say, taking Rls to consist of only SParBeta'), i.e., interpreting the variable metavariables and term metavariables of SParBeta' in arbitrary ways, i.e., essentially interpreting u and U_1, U_2, U_1', U_2' as arbitrary variable x and terms t_1, t_2, t_1', t_2', and evaluating the side-condition accordingly, we obtain exactly (ParBeta'). □

Before stating Urban et al.'s strong induction criterion, let us recall the baseline induction principle associated to Fig. 12's definition of $J_{Rls,\mathcal{T}}$ (and following directly from that definition):

$$\frac{(\mathit{TVr}\ U_1, \mathit{TVr}\ U_1') \qquad (\mathit{TVr}\ U_2, \mathit{TVr}\ U_2')}{\begin{array}{c}(\mathit{SOp}\ ap\ (\mathit{SAbs}\ u\ (\mathit{TVr}\ U_1), \mathit{TVr}\ U_2),\\ \mathit{SOp}\ sub\ (\mathit{TVr}\ U_1', \mathit{TVr}\ U_2', \mathit{SVr}\ u))\end{array}} \begin{array}{l}[((\mathit{TVr}\ U_2, \mathit{TVr}\ U_2'),\\ \ (u),\\ \ \lambda(t,t'), x.\ (x \notin FV\ t \cup FV\ t'))]\end{array}$$

Fig. 13. Schematic rule corresponding to (ParBeta'). Note that the side-condition's predicate, $\lambda(t, t'), x. (x \notin FV\ t \cup FV\ t')$, is aimed to be evaluated on term interpretations of the tuple (pair) of sterms $(\mathit{TVr}\ U_2, \mathit{TVr}\ U_2')$ and varriable interpretations of the variable metavariable u.

Thm 30. Assume Rls is a set of schematic rules over Σ and n, and $\mathcal{T} = (T, \_[\_], Vr, Abs, Op)$ is a Σ-enriched nominal set. Let and $\varphi : T^n \longrightarrow Bool$ and assume the following holds:

For all $((hyps_1, \ldots, hyps_k), conc, (side_1, \ldots, side_l)) \in Rls$, $\rho : VMVar \longrightarrow Var$ and $\delta : TMVar \longrightarrow T$, we have that

(1) $\bigwedge_{i=1}^{k} J_{Rls,\mathcal{T}}(int_{\mathcal{T}}\ \rho\ \delta\ hyps_i) \wedge \varphi\ (int_{\mathcal{T}}\ \rho\ \delta\ hyps_i)$ and

(2) $\bigwedge_{i=1}^{l} sideP_i\ (int_{\mathcal{T}}\ \rho\ \delta\ sideT_i)\ (\overline{\rho}\ sideV_i)$

imply

(3) $\varphi\ (int_{\mathcal{T}}\ \rho\ \delta\ conc)$.

Then $\forall t \in T.\ J_{Rls,\mathcal{T}}\ t \longrightarrow \varphi\ t$.

Urban et al.'s criterion, which we will describe next, is an improvement of the above that allows the variables appearing bound in the rules' conclusions to be assumed disjoint from given finite sets of variables (produced via finitely supported parameters).

We define the set of *bound variable metavariables* $BVMs\ s$ of an sterm s to consist of those variable metavariables appearing in abstraction subterms $SAbs\ u\ s$, namely:

- $BVMs\ (VVr\ u) = \emptyset$
- $BVMs\ (TVr\ U) = \emptyset$
- $BVMs\ (SAbs\ u\ s) = \{u\} \cup BVMs\ s$
- $BVMs\ (SOp\ \sigma\ (s_1, \ldots, s_{arOf\ \sigma}) = \bigcup_{i=1}^{arOf\ \sigma} BVMs\ s_i$

This is extended as expected to tuples of terms

$$BVMs\ (s_1, \ldots, s_k) = \bigcup_{i=1}^{k} BVMs\ s_i$$

and then to entire rules: If $rl = (hyps, conc, side)$ where $hyps = (hyps_1, \ldots, hyps_k)$ and $side = (side_1, \ldots, side_l)$ with each $side_i$ having the form $(sideT_i, sideV_i, sideP_i)$, then

$$BVMs\ rl = (\textstyle\bigcup_{i=1}^{k} BVMs\ hyps_i) \cup (\bigcup_{i=1}^{l} BVMs\ sideT_i) \cup BVMs\ conc$$

In short, the bound variable metavariables of a rule are those variable metavariables occurring in an abstraction inside of an sterm in that rule's hypotheses, side conditions or conclusion.

Now we are almost ready to state the Urban et al. criterion, which is based on two requirements. First, it requires that, for each rule $rl \in Rls$, equivariance holds for all the involved operators and predicates. Second, it requires, for each rule $rl \in Rls$, the following condition, formulated quite informally [Urban et al. 2007, Def. 5]: "the side-conditions $S_1\ ss_1 \wedge \ldots \wedge S_m\ ss_m$ imply that the variables in *as* are fresh for *ts* and they are distinct", where:

- *ts* is the tuple of term-like items from *rl*'s conclusion, i.e., *conc* in our notation;
- S_i are the side-condition predicates, in our notation, $sideP_i$.

As for the ss_i's mentioned above, the only way to make sense of them is as the *interpretations* of the tuples of variable metavariables and sterms appearing in the side-conditions—so, in our notation,

$$\frac{(TVr\ U_1,\ TVr\ U_1') \qquad (TVr\ U_2,\ TVr\ U_2')}{(SOp\ ap\ (SAbs\ u\ (TVr\ U_1),\ TVr\ U_2),\ \ SOp\ sub\ (TVr\ U_1',\ TVr\ U_2',\ SVr\ u))} \quad \begin{array}{l}[((),\\ \ (),\\ \ \lambda\ \_\ \_.\ True]\end{array}$$

Fig. 14. Schematic rule corresponding to (ParBeta). The side-condition is vacuous (trivially true).

the interpretations of the tuples $sideT_i$ and $sideV_i$ (and not the tuples themselves).[1] Just to have a name for it,we will call this second Urban et al. condition *vc-amenability*. Our above discussion leads to the following definition:

Def 31. A rule $rl = (hyps, conc, side)$, where $conc = (conc_1, \ldots, conc_n)$ and $side = (side_1, \ldots, side_l)$, is said to be *variable-convention- (vc-) amenable* if, for all $u \in BVMs\ rl$, $\rho : VMVar \to Var$ and $\delta : TMVar \to T$, we have that $\bigwedge_{i=1}^{l} sideP_i\ (int_{\mathcal{T}}\ \rho\ \delta\ sideT_i)\ (\overline{\rho}\ sideV_i)$ implies that $\rho\ u$ is fresh for all items in the tuple $int_{\mathcal{T}}\ \rho\ \delta\ conc$, i.e., $\bigwedge_{j=1}^{n} \rho\ u \notin Tvars\,(int_{\mathcal{T}}\ \rho\ \delta\ conc_j)$.

(As it turns out, we do not actually need any condition corresponding to the Urban et al. aforementioned distinctness requirement.)

Example 32. We can check that the schematic rule SParBeta' from Example 29 is vc-amenable. Indeed, $BVMs\,(\text{SParBeta'}) = \{u\}$, so, also expanding the definitions of the particular components of this rule and the recursive definition of $int_{\mathcal{T}}$, we see that vc-amenability amounts to the following property:

For all $\rho : VMVar \to Var$ and $\delta : TMVar \to LTerm$,
$\rho\ u \notin Tvars\,(\delta\ U_2) \cup Tvars\,(\delta\ U_2')$
implies
$\rho\ u \notin Tvars\,(Op\ ap\,(Abs\ (\rho\ u)\ (\delta\ U_1), \delta\ U_2))$ and $\rho\ u \notin Tvars\,(Op\ sub\,(\delta\ U_1', \delta\ U_2', \rho\ u))$.

Writing x for $\rho\ u$, t_i for $\delta\ U_i$ and t_i' for $\delta\ U_i'$ (where $i \in \{1,2\}$), this is equivalent to:

For all $x \in Var$ and $t_1, t_1', t_2, t_2' \in LTerm$,
$x \notin Tvars\ t_2 \cup Tvars\ t_2'$
implies
$x \notin Tvars\,(Op\ ap\,(Abs\ x\ t_1), t_2))$ and $x \notin Tvars\,(Op\ sub\,(t_1', t_2', x))$.

Furthermore, applying the definitions of *Tvars*, *Op* and *Abs* for this particular Σ-enriched nominal set, this is equivalent to:

For all $x \in Var$ and $t_1, t_1', t_2, t_2' \in LTerm$,
$x \notin FV\ t_2 \cup FV\ t_2'$
implies
$x \notin FV\ (Ap\,(Lm\ x\ t_1)\ t_2)$ and $x \notin FV\ (t_1'[t_2'/x])$.

The last is true by the properties of free variables and substitution on λ-terms, since $FV\ (Ap\,(Lm\ x\ t_1)\ t_2) = FV\ t_1 \setminus \{x\} \cup FV\ t_2$ and $FV\ (t_1'[t_2'/x]) = FV\ t_1' \setminus \{x\} \cup FV\ t_2'$.

On the other hand, the schematic rule corresponding to the rule (ParBeta) from §2 (i.e., (ParBeta') without the side-condition), shown in Fig. 14, is not vc-amenable, because its vc-amenability is equivalant to the following clearly false statement:

For all $x \in Var$ and $t_1, t_1', t_2, t_2' \in LTerm$, $x \notin FV\ (Ap\,(Lm\ x\ t_1)\ t_2)$ and $x \notin FV\ (t_1'[t_2'/x])$. □

Now we can rigorously define Urban et al.'s notion of vc-compatibility, and state their criterion:

[1]This is the main point where we must resolve the ambiguity of variables and terms versus variable metavariables and sterms from Urban et al.'s account.

Def 33. Let $\mathcal{T} = (T, \_[\_], Abs, Op)$ is a Σ-enriched nominal set, $n \in \mathbb{N}$, and Rls be a set of schematic rules over Σ and n. Then we say that the pair $(\mathcal{T}, Rls)$ is *variable-convention- (vc-) compatible* if the following two conditions hold:

- Abs, Op and all the predicates $sideP_i$ from side-conditions of the schematic rules in Rls are equivariant.
- the schematic rules in Rls are vc-amenable.

Thm 34. [Urban et al. 2007] Assume Rls is a set of schematic rules over Σ and n, and $\mathcal{T} = (T, \_[\_], Vr, Abs, Op)$ is a Σ-enriched nominal set such that $(\mathcal{T}, Rls)$ is vc-compatible.

Let P be a set and $Psupp : P \to \mathcal{P}_{\text{fin}}(Var)$. Let $\varphi : P \to T^n \to Bool$ and assume the following holds:

For all $p \in P$, $rl = ((hyps_1, \ldots, hyps_k), conc, (side_1, \ldots, side_l)) \in Rls$, $\rho : VMVar \to Var$ and $\delta : TMVar \to T$ such that $BVMs\ rl \cap Psupp\ p = \emptyset$, we have that

(1) $\bigwedge_{i=1}^{k} J_{Rls,\mathcal{T}}(int_{\mathcal{T}}\, \rho\, \delta\, hyps_i) \wedge (\forall p.\ \varphi\ p\ (int_{\mathcal{T}}\, \rho\, \delta\, hyps_i))$
and
(2) $\bigwedge_{i=1}^{l} sideP_i\ (int_{\mathcal{T}}\, \rho\, \delta\, sideT_i)\ (\overline{\rho}\ sideV_i)$
imply
(3) $\varphi\ (int_{\mathcal{T}}\, \rho\, \delta\, conc)$.

Then $\forall p \in P,\ t \in T.\ J_{Rls,\mathcal{T}}\, t \longrightarrow \varphi\ p\ t$.

Proof. We will show that the structures and assumptions of this theorem are a particular case of those of our Thm. 7. Since "$\mathcal{T}$" is already in use (denoting the fixed Σ-enriched nominal set), we will "prime" the notation for the nominal set required by Thm. 7, thus denoting it by $\mathcal{T}' = (T', \_[\_]'^{\mathcal{T}'})$. We will write $\_[\_]'$ instead of $\_[\_]'^{\mathcal{T}'}$ and $Supp'$ instead of $Supp'^{\mathcal{T}'}$.

We take $\mathcal{T}' = (T', \_[\_]')$ to be n'th power of the nominal set $(T, \_[\_])$, more precisely we define $T' = T^n$ and $(t_1, \ldots, t_n)[\_] = (t_1[\_], \ldots, t_n[\_])$. Note that $Supp'\ (t_1, \ldots, t_n) = \bigcup_{i=1}^{n} Supp\ t_i$.

We define $G : (T' \to Bool) \to (\mathcal{P}_{\text{fin}}(Var) \to T' \to Bool)$ as follows:

$$
\begin{aligned}
&G\ \varphi\ B\ t' \equiv \\
&\quad \exists rl = ((hyps_1, \ldots, hyps_k), conc, (side_1, \ldots, side_l)) \in Rls.\ \exists \rho : VMVar \to Var,\ \delta : TMVar \to T. \\
&\qquad B = Im\ \rho\ (BVMs\ rl) \ \wedge\ t' = int_{\mathcal{T}}\, \rho\, \delta\, conc \ \wedge \\
&\qquad (\textstyle\bigwedge_{j=1}^{k} \varphi\ (int_{\mathcal{T}}\, \rho\, \delta\, hyps_j)) \ \wedge\ (\bigwedge_{i=1}^{l} sideP_i\ (int_{\mathcal{T}}\, \rho\, \delta\, sideT_i)\ (\overline{\rho}\ sideV_i))
\end{aligned}
$$

Note that, if we ignore the $\mathcal{P}_{\text{fin}}(Var)$-argument B (highlighted above), G is just the operator underlying Fig. 12's inductive definition of $J_{Rls,\mathcal{T}}$. In addition, G requires that B is the interpretation (via ρ) of all the bound variable metavariables of the given rule.

We now verify for $\mathcal{T}'$ and G the hypotheses of Thm. 7:

- G is immediately monotonic (in fact, the monotonicy of G is what guarantees the correctness of $J_{Rls,\mathcal{T}}$'s definition in the first place).
- That G is equivariant follows from all the involved operators and predicates being assumed equivariant.
- For verifying the $\mathcal{T}'$-refreshability of G, we verify the stronger $\mathcal{T}'$-freshness condition (see Def. 6): by our choice of B, this condition is here equivalent to the assumed vc-amenability.

We therefore infer the conclusion of Thm. 7, which we apply for the parameter structure $(P, Psupp)$. Expanding the definition of G, this gives us exactly the desired induction principle. □

Summary. In the main paper we mention two improvements of our strong induction criterion compared to Urban et al's criterion, namely (1) the ability to cope with "native" rules of calculi such as (ParBeta) thanks to the more general condition ($\mathcal{T}$-refreshability) and (2) the more general,

$$P \parallel Q \equiv Q \parallel P \text{ (ParCommut)} \qquad (P \parallel Q) \parallel R \equiv P \parallel (P \parallel R) \text{ (ParAssoc)} \qquad P \parallel 0 \equiv P \text{ (ParZ)}$$

$$\nu(x).\,\nu(y).\,P \equiv \nu(y).\,\nu(x).\,P \text{ (NuCommut)} \qquad \nu(x).\,0 \equiv 0 \text{ (NuZero)}$$

$$\nu(x).\,(P \parallel Q) \equiv (\nu(x).\,P) \parallel Q \;\begin{matrix}\text{(NuPar)}\\ [x \notin FV\,Q]\end{matrix} \qquad !P \equiv P \parallel !P \text{ (Repl)}$$

$$P \equiv P \text{ (Refl)} \qquad \frac{P \equiv Q}{Q \equiv P}\text{ (Sym)} \qquad \frac{P \equiv Q \quad Q \equiv R}{Q \equiv R}\text{ (Trans)}$$

Structural rules

$$(\overline{a}\,x.\,P) \parallel (a(y).\,Q) \Longrightarrow P \parallel (Q[y/x]) \text{ (Com)} \qquad \frac{P \Longrightarrow Q}{P \parallel R \Longrightarrow Q \parallel R}\text{ (ParCong)}$$

$$\frac{P \Longrightarrow Q}{\nu(x).\,P \Longrightarrow \nu(x).\,Q}\text{ (NuCong)} \qquad \frac{P \equiv P' \quad P \Longrightarrow Q \quad Q \equiv Q'}{P' \Longrightarrow Q'}\text{ (Compat)}$$

Operational rules (reduction semantics modulo $\equiv$)

Fig. 15. Variant of π-calculus based on structural congruence

semantic nature, based on Knaster-Tarski, which allows more flexible rules, not having to fit a given format. Above we showed how our criterion implies theirs. Having to define their criterion rigorously incurs a significant amount of technical details, which we believe further advocates for the comparative elegance of our semantic approach. On the other hand, admittedly a rule-format based criterion seems more straightforward to implement.

B STRONG RULE INDUCTION FOR A π-CALCULUS WITH STRUCTURAL RULES

Next we show how our strong rule induction criterion applies to one of the standard presentations of π-caclulus [Milner et al. 1992; Sangiorgi and Walker 2001], namely one that:

- first defines some structural rules, via an inductively defined equivalence relation $\equiv$ on processes;
- then defines a reduction semantics $\Longrightarrow$ on processes that operates modulo $\equiv$.

Fig. 15 shows the inductive definitions of these two relations. Thm. 7 instantiates to both of them and gives the following strong rule induction principles for them. The verification of the theorem's hypotheses proceeds again seamlessly along our §6's heuristic.

Prop 35. Let $(P, Psupp : P \to \mathcal{P}_{\text{fin}}(Var))$ be a parameter structure. Let $\varphi : P \to Proc \to Proc \to Bool$ and assume the following hold:

- [cases different from (|NuCommut|), (|NuZero|) and (|NuPar|) omitted, as they don't involve binders][2]
- (|NuCommut|): $\forall p, x, y, P.\ x, y \notin Psupp\ p \longrightarrow \varphi\ p\ (\nu(x).\,\nu(y).\,P)\ (\nu(y).\,\nu(x).\,P)$
- (|NuZero|): $\forall p, x.\ x \notin Psupp\ p \longrightarrow \varphi\ p\ (\nu(x).\,0)\ 0$
- (|NuPar|): $\forall p, x, P.\ x \notin Psupp\ p \wedge x \notin FV\,Q \longrightarrow \varphi\ p\ (\nu(x).\,(P \parallel Q))\ ((\nu(x).\,P) \parallel Q)$

Then $\forall p, P, P'.\ P \equiv P \longrightarrow \varphi\ p\ P\ P'$.

[2]Here and elsewhere (inclding throughout the paper): We omit these cases because they do not exhibit anything new compared to standard induction. But this is not to suggest that the rules corresponding to these cases are completely irrelevant for the conditions that need to be verified in order for Thm. 23 to apply—for example, their building blocks must still be equivariant.

Prop 36. Let $(P, Psupp : P \to \mathcal{P}_{\text{fin}}(Var))$ be a parameter structure. Let $\varphi : P \to Proc \to Proc \to Bool$ and assume the following hold:
- [cases different from (|Com|) and (|NuCong|) omitted, as they don't involve binders]
- (|Com|): $\forall p, a, x, y, P.\ y \notin Psupp\ p \cup \{a, x\} \cup FV\ P \longrightarrow \varphi\ p\ (\overline{a}\, x. P) \parallel (a(y). Q))\ (P \parallel (Q[y/x]))$
- (|NuCong|): $\forall p, x, P, Q.\ x \notin Psupp\ p \wedge (P \Longrightarrow Q) \wedge (\forall p'. \varphi\ p'\ P\ Q) \longrightarrow \varphi\ p\ (\nu(x). P)\ (\nu(x). Q)$

Then $\forall p, P, P'. (P \Longrightarrow P') \longrightarrow \varphi\ p\ P\ P'$.

Since $\equiv$ participates in the definition of $\Longrightarrow$, its equivariance is required in order to instantiate Thm. 7 to $\Longrightarrow$ (yielding Prop. 36).

Similarly to other situations discussed in the main paper, again Prop. 36 shows some improvements compared to prior state of the art. Namely, it allows us to assume not only $y \notin Psupp\ p$, but also $y \notin \{a, x\}$ and $y \notin FV\ P$, whereas, in order to apply, the [Urban et al. 2007] criterion would instead require that $y \notin \{a, x\}$ and $y \notin FV\ P$ be added as side-conditions to the rule (Com).

C MORE DETAILS AND PROOFS ABOUT κ-LS-NOMINAL SETS

In this section we give more details about κ-LS-nominal sets, including the connection with nominal sets (§C.1), and the closure properties enjoyed by the κ-LS-nominal sets (§C.2) to which we alluded at the end of §9.2 (in connection with goal (G3)).

C.1 Connection with nominal sets

First, the straightforward fact that κ-LS-nominal sets generalize nominal sets:

Lemma 37. $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ be an $\aleph_0$-LS-nominal set (so $\kappa = \aleph_0$). Then $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ is a nominal set, and for all $a \in A$, $Supp^{\mathcal{A}}\ a$ is a finite supporting set for a (in the nominal-set sense).

Proof. This follows immediately for the definition, since "$\aleph_0$-small" means "finite". □

Note that, when moving from nominal sets to κ-LS-nominal sets, only the replacement of finiteness by κ-smallness is a generalization. The other variation, namely the consideration of a "loose" supporting-set operator, is more of a particularization: It refers to choosing and making explicit in the structure some data that was already available in the notion of nominal set (and, as explained in §9.2, its role is to calibrate/facilitate the generalization from finiteness to κ-smallness). This is situation is reflected in an adjunction between the categories of these two structures for $\kappa = \aleph_0$, which we describe next.

We let $\underline{Nom}$ [Pitts 2013] be the category whose objects are the nominal sets, and whose morphisms, say between $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ and $\mathcal{B} = (B, \_[\_]^{\mathcal{B}})$, are permutation-commuting (i.e., equivariant) functions between their carrier sets $f : A \to B$, in that the following holds:

$f\ (a[\sigma]^{\mathcal{A}}) = (f\ a)[\sigma]^{\mathcal{B}}$ for all $a \in A$ and $\sigma \in Perm$

Moreover, for each infinite κ, we let $\underline{\mathcal{LSNom}}_\kappa$ be the category whose objects are κ-LS-nominal sets, and whose morphisms, say between $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ and $\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, Supp^{\mathcal{B}})$, are functions between the carrier sets $f : A \to B$ that are permutation-commuting and support-preserving, in that the following hold:

- $f\ (a[\sigma]^{\mathcal{A}}) = (f\ a)[\sigma]^{\mathcal{B}}$ for all $a \in A$ and $\sigma \in Perm$
- $Supp^{\mathcal{B}}(f\ a) \subseteq Supp^{\mathcal{A}}\ a$ for all $a \in A$

(It is easy to see that $\underline{\mathcal{LSNom}}_\kappa$ forms indeed a category.)

Now we define the following functors $F : \underline{\mathcal{LSNom}}_{\aleph_0} \to \underline{Nom}$ and $G : \underline{Nom} \to \underline{\mathcal{LSNom}}_{\aleph_0}$.

- On objects, for $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$, we take $F\,\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$; on morphisms, we take $F\ f = f$.

- On objects, for $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$, we take $G\,\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, \mathit{Supp}^{\mathcal{A}})$, where $\mathit{Supp}^{\mathcal{A}}$ is the standard support operator (giving the least supporting set) on the nominal set $\mathcal{A}$; on morphisms, we take $G\,f = f$.

That F is well-defined is straightforward.

That G is well-defined on objects amounts to nominal sets satisfying the property of semi-naturality of support w.r.t. permutation: $\mathit{Supp}^{\mathcal{A}}\,(a[\sigma]) \subseteq \mathit{Im}\,\sigma\,(\mathit{Supp}^{\mathcal{A}}\,a)$ for all $a \in A$ and $\sigma \in \mathit{Perm}$. That G is well-defined on morphisms amounts to the property that equivariant functions between nominal sets also preserve the support (in the above sense, i.e., of not introducing new atoms), or, equivalently, preserve the freshness predicate. Both of these are well-known properties of nominal sets [Pitts 2013]. (And of course the functoriality of both F and G is straightforward because they are the identity on morphisms.)

Prop 38. The functors F and G form an adjunction $F \dashv G$ between $\underline{\mathcal{LSNom}}_{\aleph_0}$ and $\underline{\mathcal{Nom}}$. (So F is the left adjoint.)[3]

Proof. The essence of this adjunction is that, given an $\aleph_0$-LS-nominal-set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, \mathit{Supp}^{\mathcal{A}})$, a nominal set $\mathcal{B} = (B, \_[\_]^{\mathcal{B}})$, and a function $f : A \to B$, the following statements are equivalent:

- f is a κ_0-LS-nominal-set morphism between $\mathcal{A}$ and $G\,\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, \mathit{Supp}^{\mathcal{B}})$ (where $\mathit{Supp}^{\mathcal{B}}$ is therefore the standard support operator of the nominal set $\mathcal{B}$).
- f is a nominal-set morphism between $F\,\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ and $\mathcal{B}$.

The left-to-right implication is immediate.

For the right-to-left implication, assume that f is a nominal-set morphism between $(A, \_[\_]^{\mathcal{A}})$ and $\mathcal{B}$. Let $\mathit{Supp}'^{\mathcal{A}}$ be the standard support operator of the nominal set $\mathcal{A}$ (returning the least supporting sets). $\mathit{Supp}^{\mathcal{A}}$ can of course be different from $\mathit{Supp}'^{\mathcal{A}}$, but since the former also returns some supporting sets, we have $\mathit{Supp}'^{\mathcal{A}}\;a \subseteq \mathit{Supp}^{\mathcal{A}}\;a$ for all $a \in A$. Moreover, since f is a nominal-set morphism, we know that it preserves the standard support operators, meaning $\mathit{Supp}^{\mathcal{B}}\,(f\;a) \subseteq \mathit{Supp}'^{\mathcal{A}}\;a \subseteq \mathit{Supp}^{\mathcal{A}}\;a$ for all $a \in A$, which makes f a nominal-set morphism between $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, \mathit{Supp}^{\mathcal{A}})$ and $(B, \_[\_]^{\mathcal{B}}, \mathit{Supp}^{\mathcal{B}})$, as desired. □

C.2 Closure properties

We will express the closure properties using the notion of κ-natural functor. These are inspired by Traytel et al.'s bounded natural functors (BNFs) [Traytel et al. 2012] but have fewer restrictions (e.g., they are nor required to preserve weak pullbacks).

Def 39. Given $n \in \mathbb{N}$ and a cardinal κ, an n-ary *κ-natural functor* is a triple $(G, \mathit{Gmap}, (\mathit{Gset}^i)_{i\in\{1,\ldots,n\}})$ where (G, Gmap) is an n-ary endofunctor on the category of sets and functions, each Gset^i is a natural transformation between the i'th component of (G, Gmap) and the κ-bounded powerset functor, and Gmap satisfies the Gset-congruence property:

In more detail, each Gset^i is a family $(\mathit{Gset}^i_{\overline{A}})_{\overline{A}\in \mathit{Set}^n}$ where $\mathit{Gset}^i_{(A_1,\ldots,A_n)} : G(A_1,\ldots,A_n) \to \mathcal{P}_{<\kappa}(A_i)$ such that the following properties hold:

- Naturality: For any tuples of sets $\overline{A} = (A_1,\ldots,A_n)$ and $\overline{B} = (B_1,\ldots,B_n)$, and of functions $\overline{f} = (f_1 : A_1 \to B_1,\ldots,f_n : A_n \to B_n)$, it holds that $(\mathit{Im}\,f_i) \circ \mathit{Gset}^i_{\overline{A}} = \mathit{Gset}^i_{\overline{B}} \circ \mathit{Gmap}\,\overline{f}$.
- Congruence: For any tuples of sets $\overline{A} = (A_1,\ldots,A_n)$ and $\overline{B} = (B_1,\ldots,B_n)$, and of functions $\overline{f} = (f_1 : A_1 \to B_1,\ldots,f_n : A_n \to B_n)$ and $\overline{g} = (g_1 : A_1 \to B_1,\ldots,g_n : A_n \to B_n)$, and

[3] The fact that the forgetful functor F turns out to be a *left* adjoint is a consequence of the direction in which preservation of the support operator is formulated as an inclusion: $\mathit{Supp}^{\mathcal{B}}(f\;a) \subseteq \mathit{Supp}^{\mathcal{A}}\;a$, and not $\mathit{Supp}^{\mathcal{A}}\;a \subseteq \mathit{Supp}^{\mathcal{B}}(f\;a)$. The former is indeed the correct direction, because this is the one holding for nominal sets.

for any $a \in G(A_1, \ldots, A_n)$, if $\forall i \in \{1, \ldots, n\}.\ \forall b \in Gset^i_{\overline{A}}\ a.\ f_i\ b = g_i\ b$ then $Gmap\,\overline{f}\,a = Gmap\,\overline{g}\,a$.

κ-natural functors form a very comprehensive class of functors. It includes all the bounded natural functors [Traytel et al. 2012], hence also all container-type functors [Abbott et al. 2005] such as sums, products, lists, streams, trees of various kinds, etc., as well bounded sets, multisets, etc.

Next, we will show that that our κ-LS-nominal sets are closed under the application of such functors, meaning that whenever we have κ-LS-nominal set structures on the set arguments A_i to such a functor G, we have a natural κ-LS-nominal set structure on $G(A_1, \ldots, A_n)$ as well. In particular, this gives us sums and products of κ-LS-nominal sets.

The utility of this result for formal proof engineering, in particular for the application of our LS-nominal-set based strong rule induction criteria (Thms. 20 and 23) is that we have a standard and automatic way to endow with LS-nominal-set structure any datatype whose basic building blocks are already LS-nominal sets—and these are the typical domains of the inductively defined predicates of interest, in other words Thms. 20 and 23 can be seamlessly fed with the necessary LS-nominal set structures. The great utility of such closure properties for nominal sets [Pitts 2006] is illustrated by the success of the Nominal Isabelle package [Urban and Tasson 2005].

Prop 40. Assume that κ is a regular cardinal[4] and $\kappa' < \kappa$. Then LS-nominal sets are closed under the applications of n-ary κ'-*natural* functors (for any n) on the category of sets.

Proof. Let $\mathcal{A}_i = (A_i, \_[\_]^{\mathcal{A}_i}, Supp^{\mathcal{A}_i})$ for $i \in \{1, \ldots, n\}$ be n permutative sets and let $(G, Gmap, Gset)$ be an n-ary κ'-natural functor. We define the following structure $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ on the carrier set $A = G(A_1, \ldots, A_n)$:

- $\_[\_]^{\mathcal{A}}$ is defined by $\_[\sigma]^{\mathcal{A}} = Gmap\,(\_[\sigma]^{\mathcal{A}_1}, \ldots\ \_[\sigma]^{\mathcal{A}_n})$
- $Supp^{\mathcal{A}} : A \to \mathcal{P}_{<\kappa}(Var)$ is defined by $Supp^{\mathcal{A}}\ a = \bigcup_{i=1}^{n} \bigcup_{u \in Gset^i\ a} Supp^{\mathcal{A}_i}\ u$.

That $Supp^{\mathcal{A}}$ is well defined, i.e., that $|Supp^{\mathcal{A}}\ a| < \kappa$ for all a, follows from $|Gset^i\ s| < k'$, $|Svars_i\ u| < \kappa$, $\kappa' < \kappa$ and κ being regular.

The κ-pre-LS-nominal set properties of $\mathcal{A}$ follow form the corresponding properties of the $\mathcal{A}_i$'s and the functoriality of $Gmap$. The fact that, for all $a \in G(A_1, \ldots, A_n)$, $Supp^{\mathcal{A}}\ a$ is a supporting set for a, follows from the corresponding properties of each $Supp^{\mathcal{A}_i}$ and the congruence property of $Gmap$ w.r.t. $Gset$. Finally, semi-naturality of $Supp^{\mathcal{A}}$ follows from the semi-naturality of each $Supp^{\mathcal{A}_i}$ and the naturality of each $Gset^i$. □

D TERMS WITH BINDINGS ORGANIZED INTO ABSTRACT DATATYPES

This paper's results were concerned with strong rule induction, and are datatype-agnostic: They work with predicates defined inductively on any nominal set, or more generally any κ-LS-nominal set. But our examples of course involve specific (κ-LS-)nominal sets, which are always extensions or variations of (possibly infinitary) sets of terms with bindings of some sort. Moreover, verifying the assumptions of our theorems typically requires basic properties of terms with bindings. In what follows, we describe the necessary background for datatypes of terms with bindings. More precisely, for the signatures of finitary and infinitary λ-calculus and of the π-calculus, we describe properties of the corresponding terms over this signature, considered modulo alpha-equivalence.

We will take an *abstract datatype* view. Namely, we will not show any concrete construction of terms as alpha-equivalence classes—several equivalent constructions are possible, e.g., [Barendregt 1985; Pitts 2006; Urban 2008]. Instead, we list properties that characterize the datatypes of terms

[4] Regular cardinals include $\aleph_0$ and all infinite successor cardinals, in particular $\aleph_1$, the first uncountable cardinal.

and their basic operators (namely the constructors, permutation and free-variable operators) as an abstract datatype, i.e., up to structure-preserving isomorphism.

For each such abstract datatype, we will have a recursion principle, allowing one to define functions recursively on that type. There are several choices for such a recursor—Popescu [2024] gives an overview. Our choice is a variation of the one described by Norrish [Norrish 2004] for the syntax of λ-calculus. More precisely, we use the recursor from Blanchette et al. [Blanchette et al. 2019], who generalize Norrish's recursor to an arbitrary (possibly infinitary) syntax with bindings. These are similar to recursors developed by Gabbay and Pitts [Gabbay and Pitts 2002], Pitts [Pitts 2006] and Urban and Berghofer [Urban and Berghofer 2006] in the context of nominal logic. While Blanchette et al. describe the recursor in functorial terminology (employing so-called *map-restricted bounded natural functors*, *MRBNFs*), we here reformulate it using our concepts and notations (employing a variation of κ-LS-nominal sets).

D.1 Finitary λ-terms as an abstract datatype

In this subsection, *Var* is a countable set of variables and *Perm* is the set of permutations on *Var*, meaning here bijections of finite support. Recall from §2 that (finitary) λ-terms, forming the set *LTerm*, are generated by the constructors $Vr : Var \to LTerm$, $Ap : LTerm \to LTerm \to LTerm$ and $Lm : Var \to LTerm \to LTerm$. Of these constructors, *Vr* and *Ap* are free—whereas *Lm* is not, but a "quasi-injectivity" / "injectivity up to renaming" property holds for it (Lemma 41(6) below). In addition to the constructors, we also have the free-variable operator $FV : LTerm \to \mathcal{P}_{\mathsf{fin}}(Var)$ and permutation operator $\_[\_] : LTerm \to Perm \to LTerm$.

Lemma 41. (Distinctness and (quasi-)injectivity of the constructors) The following hold:
(1) $Vr\ x \neq Ap\ t_1\ t_2$;
(2) $Vr\ x \neq Lm\ x'\ t$;
(3) $Ap\ t_1\ t_2 \neq Lm\ x\ t$;
(4) $Vr\ x = Vr\ x'$ iff $x = x'$;
(5) $Ap\ t_1\ t_2 = Ap\ t_1'\ t_2'$ iff $t_1 = t_1'$ and $t_2 = t_2'$;
(6) $Lm\ x\ t = Lm\ x'\ t'$ iff there exists y that is fresh for x, x', t, t' (i.e., $y \notin \{x, x'\} \cup FV\ t \cup FV\ t'$) such that $t[y \leftrightarrow x] = t'[y \leftrightarrow x']$.

Lemma 42. (Equivariance of the constructors) The following hold, assuming $\sigma \in Perm$:
(1) $(Vr\ y)[\sigma] = Vr\ (\sigma\ y)$;
(2) $(Ap\ t_1\ t_2)[\sigma] = Ap\ (t_1[\sigma])\ (t_2[\sigma])$;
(3) $(Lm\ x\ t)[\sigma] = Lm\ (\sigma\ x)\ (t[\sigma])$.

Lemma 43. (Free variables versus constructors) The following hold:
(1) $FV\ (Vr\ y) = \{y\}$;
(2) $FV\ (Ap\ t_1\ t_2) = FV\ t_1 \cup FV\ t_2$;
(3) $FV\ (Lm\ x\ t) = FV\ t \setminus \{x\}$.

Lemma 44. (Structural induction) Assume $\varphi : LTerm \to Bool$ is a predicate such that the following hold:

- $\forall x.\ \varphi\ (Vr\ x)$;
- $\forall t_1, t_2.\ \varphi\ t_1 \wedge \varphi\ t_2 \longrightarrow \varphi\ (Ap\ t_1\ t_2)$;
- $\forall x, t.\ \varphi\ t \longrightarrow \varphi\ (Lm\ x\ t)$.

Then $\forall t.\ \varphi\ t$.

In the above lemmas, the only place where the fact that we work not with entirely free terms but with terms quotiented to alpha is visible, is Lemma 41(6).

Note that the structural induction principle (Lemma 44) means that the vacuously true predicate on terms is the same as the predicate K defined inductively by the following clauses:

$$K\ (Vr\ x)\ \text{(Kr)} \qquad \frac{K\ t_1 \quad K\ t_2}{K\ (Ap\ t_1\ t_2)}\ \text{(Ap)} \qquad \frac{K\ t}{K\ (Lm\ x\ t)}\ \text{(Lm)}$$

Thanks to this observation, our strong rule induction criterion (Thm. 7) applies, yielding the following:

Lemma 45. (Strong structural induction) Assume $(P, Psupp : P \to \mathcal{P}_{\text{fin}}(Var))$ is a parameter structure and $\varphi : P \to LTerm \to Bool$ is a predicate such that the following hold:

- $\forall p, x.\ \varphi\ p\ (Vr\ x)$;
- $\forall p, t_1, t_2.\ (\forall q.\ \varphi\ q\ t_1)\ \wedge\ (\forall q.\ \varphi\ q\ t_2) \longrightarrow \varphi\ p\ (Ap\ t_1\ t_2)$;
- $\forall p, x, t.\ x \notin Psupp\ p \wedge (\forall q.\ \varphi\ q\ t) \longrightarrow \varphi\ p\ (Lm\ x\ t)$.

Then $\forall p, t.\ \varphi\ p\ t$.

So strong structural induction is a particular case of strong rule induction, although the former is typically established independently at the time when the datatype of terms is defined (e.g., [Pitts 2006], [Urban 2008], [Blanchette et al. 2019]).

λ-terms form a standard example of a nominal set, in particular they form an $\aleph_0$-LS-nominal set. In order to *characterize uniquely* the λ-terms among the structures equipped with constructor and LS-nominal set operators, we will use a notion that is weaker than nominal sets, and even weaker than $\aleph_0$-LS-nominal sets.

Just for the next definition, we will stop assuming that Var is countable, but perform the definition under the more general assumption that the cardinality of Var is an infinite regular cardinal κ.

Remember (from Def. 19) that κ-LS-nominal sets are tuples $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ where $(A, \_[\_]^{\mathcal{A}})$ is a κ-pre-nominal set,

$Supp^{\mathcal{A}}$ returns supported sets and $Supp^{\mathcal{A}}$ is semi-natural. Now we introduce κ-quasi-LS-nominal sets by simply removing the semi-naturality assumption.

Def 46. A *κ-quasi-LS-nominal set* (*κ-QLS-nominal set* for short) is a triple $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ where A is a set, and $\_[\_]^{\mathcal{A}} : A \to Perm \to A$ and $Supp^{\mathcal{A}} : A \to \mathcal{P}_{<\kappa}(Var)$ are such that:
- $(A, \_[\_]^{\mathcal{A}})$ is a κ-pre-nominal set;
- $Supp^{\mathcal{A}}$ returns supported sets w.r.t. $\_[\_]^{\mathcal{A}}$, i.e., $(\forall x \in Supp^{\mathcal{A}}.\ \sigma\ x = x)$ implies $a[\sigma]^{\mathcal{A}}$ for all a and σ;

Note that the concept of equivariance also makes sense for κ-QLS-nominal sets.

Now we are back to assuming Var countable, i.e., that $\kappa = \aleph_0$. So a nominal set is in particular an $\aleph_0$-LS-nominal set, which is in particular an $\aleph_0$-QLS-nominal set. In the rest of this subsection, we will omit the "$\aleph_0$" qualification, thus simply writing "LS-nominal set" and "QLS-nominal set".

Def 47. A *λ-enriched QLS-nominal set* is a tuple $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, Vr^{\mathcal{A}}, Lm^{\mathcal{A}}, Ap^{\mathcal{A}})$ such that $(A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ is a QLS-nominal set, and $Vr^{\mathcal{A}} : Var \to A$, $Ap^{\mathcal{A}} : A \to A \to A$ and $Lm^{\mathcal{A}} : Var \to A \to A$ are operators, such that the following hold:

- $Vr^{\mathcal{A}}$, $Ap^{\mathcal{A}}$ and $Lm^{\mathcal{A}}$ are equivariant;
- $Supp^{\mathcal{A}}\ (Vr^{\mathcal{A}}\ x) \subseteq \{x\}$ for all $x \in Var$;
- $Supp^{\mathcal{A}}\ (Ap^{\mathcal{A}}\ a_1\ a_2) \subseteq Supp^{\mathcal{A}}\ a_1 \cup Supp^{\mathcal{A}}\ a_2$ for all $a_1, a_2 \in A$;
- $Supp^{\mathcal{A}}\ (Lm^{\mathcal{A}}\ x\ a) \subseteq Supp^{\mathcal{A}}\ a \setminus \{x\}$ for all $x \in Var$ and $a \in A$.

Thus, λ-enriched QLS-nominal sets are structures that emulate λ-terms to a certain degree. And indeed, the structure $\mathcal{LT}erm = (LTerm, \_[\_], FV, Vr, Lm, Ap)$ is the primary example of a λ-enriched QLS-nominal set.

Def 48. Given two *λ-enriched QLS-nominal sets* $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, Vr^{\mathcal{A}}, Lm^{\mathcal{A}}, Ap^{\mathcal{A}})$ and $\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, Supp^{\mathcal{B}}, Vr^{\mathcal{B}}, Lm^{\mathcal{B}}, Ap^{\mathcal{B}})$, a morphism between $\mathcal{A}$ and $\mathcal{B}$ is a function $h : A \to B$ that commutes or (in the case of the variable operators) sub-commutes with the operators, in the following sense:

(1) $h\,(a[\sigma]^{\mathcal{A}}) = (h\ a)[\sigma]^{\mathcal{B}}$ for all $\sigma \in Perm$ and $a \in A$;
(2) $Supp^{\mathcal{B}}\ (h\ a) \subseteq Supp^{\mathcal{A}}\ a$ for all $a \in A$;
(3) $h\,(Vr^{\mathcal{A}}\ x) = Vr^{\mathcal{B}}\ x$ for all $x \in Var$;
(4) $h\,(Ap^{\mathcal{A}}\ a_1\ a_2) = Ap^{\mathcal{B}}\ (h\ a_1)\ (h\ a_2)$ for all $a_1, a_2 \in A$;
(5) $h\,(Lm^{\mathcal{A}}\ x\ a) = Lm^{\mathcal{B}}\ x\ (h\ a)$ for all $x \in Var$ and $a \in A$.

The following recursion principle holds for λ-terms, which also characterizes the structure $\mathcal{LT}erm$ uniquely up to isomorphism:

Prop 49. $\mathcal{LT}erm$ is initial in the category of λ-enriched QLS-nominal sets. More explicitly, for any λ-enriched QLS-nominal set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, Vr^{\mathcal{A}}, Lm^{\mathcal{A}}, Ap^{\mathcal{A}})$, there exists a unique morphism from $\mathcal{LT}erm$ to $\mathcal{A}$, i.e., a function $h : LTerm \to A$ satisfying the following properties:

(1) $h\,(t[\sigma]) = (h\ t)[\sigma]^{\mathcal{A}}$ for all $\sigma \in Perm$ and $t \in LTerm$;
(2) $Supp^{\mathcal{A}}\ (h\ t) \subseteq FV\ t$ for all $t \in LTerm$;
(3) $h\,(Vr\ x) = Vr^{\mathcal{A}}\ x$ for all $x \in Var$;
(4) $h\,(Ap\ t_1\ t_2) = Ap^{\mathcal{A}}\ (h\ t_1)\ (h\ t_2)$ for all $t_1, t_2 \in LTerm$;
(5) $h\,(Lm\ x\ t) = Lm^{\mathcal{A}}\ x\ (h\ t)$ for all $x \in Var$ and $t \in LTerm$.

Of the properties (1)–(5) above, only (3)–(5) correspond to what is usually called a recursive definition, because they show how h behaves recursively on the constructors. On the other hand, (1) and (2) are additional properties of h, showing how it (sub)commutes with mapping and free-variables, which here act as "recursion-helping" operators.

D.2 Infinitary λ-terms as an abstract datatype

In this subsection, $iVar$ is a set of variables of cardinality $\aleph_1$, and $Perm = Perm_{\aleph_1}$ is the set of $\aleph_1$-permutations on Var, meaning here bijections of countable support. (Differently from the main paper, we write $iVar$ rather than Var, to avoid confusion with the countable set Var that we use for the finitary λ-calculus. Again differently from the main paper, we write $Perm$ instead of $Perm_{\aleph_1}$.)

Recall from §9.3 that infinitary λ-terms (iterms), forming the set $ILTerm$, are generated by the constructors $iVr : iVar \to ILTerm$, $iAp : ILTerm \to ILTerm^{\infty} \to ILTerm$ and $iLm : iVar^{\infty,\neq} \to ILTerm \to ILTerm$. Of these constructors, iVr and iAp are free and iLm is not. In addition to the constructors, we also have the free-variable operator $FV : ILTerm \to \mathcal{P}_{\text{countable}}(Var)$ and permutation operator $\_[\_] : ILTerm \to Perm \to ILTerm$.

We will also use the *map*, *lift* and *set* operators for streams; recall that these operators map a function and universally extend a predicate componentwise from elements to streams, and take the elements appearing in a stream, respectively: $(map\ \sigma\ as)_i = as_i$, $lift\ \varphi\ as = (\forall i \in \mathbb{N}.\ \varphi\ as_i)$, and $set\ as = \{as_i \mid i \in \mathbb{N}\}$.

Lemma 50. (Distinctness and (quasi-)injectivity of the constructors) The following hold, assuming $xs, xs' \in iVar^{\infty,\neq}$:
(1) $iVr\ x \neq iAp\ t\ ts$;
(2) $iVr\ x \neq iLm\ xs\ t$;
(3) $iAp\ t\ ts \neq iLm\ xs\ t'$;
(4) $iVr\ x = iVr\ x'$ iff $x = x'$;
(5) $iAp\ t\ ts = iAp\ t'\ ts'$ iff $t = t'$ and $ts = ts'$;

(6) $iLm\ xs\ t = iLm\ xs'\ t'$ iff there exists ys that is fresh for xs, xs', t, t' (i.e., $set\ ys \cap (set\ xs \cup set\ xs' \cup FV\ t \cup FV\ t') = \emptyset$) such that $t[ys \leftrightarrow xs] = t'[ys \leftrightarrow xs']$.

Above, given any $xs, ys \in iVar^{\infty,\neq}$ such that $set\ ys \cap set\ xs = \emptyset$, $ys \leftrightarrow xs$ denotes the permutation that takes each xs_i to ys_i and each ys_i to xs_i. (This is well-defined because the streams xs and ys are nonrepetitive and disjoint.)[5]

Lemma 51. (Equivariance of the constructors) The following hold, assuming $\sigma \in Perm$ and $xs \in iVar^{\infty,\neq}$:
(1) $(iVr\ x)[\sigma] = iVr\ (\sigma\ x)$;
(2) $(iAp\ t\ ts)[\sigma] = iAp\ (t[\sigma])\ (map\ (\_[\sigma])\ ts)$;
(3) $(iLm\ xs\ t)[\sigma] = iLm\ (map\ \sigma\ xs)\ (t[\sigma])$.

Lemma 52. (Free variables versus constructors) The following hold, assuming $xs \in iVar^{\infty,\neq}$:
(1) $FV\ (iVr\ x) = \{x\}$;
(2) $FV\ (iAp\ t\ ts) = FV\ t \cup \bigcup_{t' \in set\ ts} FV\ t'$;
(3) $FV\ (iLm\ xs\ t) = FV\ t \setminus set\ xs$.

Lemma 53. (Structural induction) Assume $\varphi : ILTerm \to Bool$ is a predicate such that the following hold:

- $\forall x.\ \varphi\ (iVr\ x)$;
- $\forall t, ts.\ \varphi\ t \wedge lift\ \varphi\ ts \longrightarrow \varphi\ (iAp\ t\ ts)$;
- $\forall xs, t.\ \varphi\ t \longrightarrow \varphi\ (iLm\ xs\ t)$.

Then $\forall t.\ \varphi\ t$.

Lemma 54. (Strong structural induction, can be obtained from plain structural induction using Thm. 20) Assume $(P, Psupp : P \to \mathcal{P}_{\text{countable}}(iVar))$ is a parameter structure and $\varphi : P \to ILTerm \to Bool$ is a predicate such that the following hold:

- $\forall p, x.\ \varphi\ p\ (iVr\ x)$;
- $\forall p, t, ts.\ (\forall q.\ \varphi\ q\ t) \wedge lift\ (\lambda t'.\ \forall q.\ \varphi\ q\ t')\ ts$
  $\longrightarrow \varphi\ p\ (iAp\ t\ ts)$;
- $\forall p, xs, t.\ set\ xs \cap Psupp\ p = \emptyset \wedge (\forall q.\ \varphi\ q\ t)$
  $\longrightarrow \varphi\ p\ (iLm\ xs\ t)$.

Then $\forall p, t.\ \varphi\ p\ t$.

Next we describe the corresponding instance of the recursor from [Blanchette et al. 2019].

Def 55. An *iλ-enriched QLS-nominal set* is a tuple $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, iVr^{\mathcal{A}}, iLm^{\mathcal{A}}, iAp^{\mathcal{A}})$ such that $(A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ is a QLS-nominal set (over $iVar$), and $iVr^{\mathcal{A}} : iVar \to A$, $iAp^{\mathcal{A}} : A \to A^{\infty} \to A$ and $iLm^{\mathcal{A}} : iVar^{\infty,\neq} \to A \to A$ are operators, such that the following hold:

- $iVr^{\mathcal{A}}$, $iAp^{\mathcal{A}}$ and $iLm^{\mathcal{A}}$ are equivariant;
- $Supp^{\mathcal{A}}\ (iVr^{\mathcal{A}}\ x) \subseteq \{x\}$ for all $x \in iVar$;
- $Supp^{\mathcal{A}}\ (iAp^{\mathcal{A}}\ a\ ss) \subseteq Supp^{\mathcal{A}}\ a \cup \bigcup_{a' \in set\ ss} Supp^{\mathcal{A}}\ a'$ for all $a \in A$ and $as \in A^{\infty}$;
- $Supp^{\mathcal{A}}\ (iLm^{\mathcal{A}}\ xs\ a) \subseteq Supp^{\mathcal{A}}\ a \setminus set\ xs$ for all $xs \in iVar^{\infty,\neq}$ and $a \in A$.

Def 56. Given two *iλ-enriched QLS-nominal sets* $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, iVr^{\mathcal{A}}, iLm^{\mathcal{A}}, iAp^{\mathcal{A}})$ and $\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, Supp^{\mathcal{B}}, iVr^{\mathcal{B}}, iLm^{\mathcal{B}}, iAp^{\mathcal{B}})$, a morphism between $\mathcal{A}$ and $\mathcal{B}$ is a function $h : A \to B$ that commutes or sub-commutes with the operators, in the following sense:

[5] Note that permutations of sufficiently large core, here countably infinite, are needed in the very statement of fundamental properties of abstractions. This is another reason why we believe that considering infinitary permutations is important when reasoning about infinitary syntax—here, specifically we need the size of the permutations to match the number of variables that can be simultaneously bound. (See also Remark 18 from the main paper.)

(1) $h\,(a[\sigma]^{\mathcal{A}}) = (h\;a)[\sigma]^{\mathcal{B}}$ for all $\sigma \in \mathit{Perm}$ and $a \in A$;
(2) $\mathit{Supp}^{\mathcal{B}}\;(h\;a) \subseteq \mathit{Supp}^{\mathcal{A}}\;a$ for all $a \in A$;
(3) $h\,(\mathit{iVr}^{\mathcal{A}}\;x) = \mathit{iVr}^{\mathcal{B}}\;x$ for all $x \in \mathit{iVar}$;
(4) $h\,(\mathit{iAp}^{\mathcal{A}}\;a\;ss) = \mathit{iAp}^{\mathcal{B}}\;(h\;a)\;(\mathit{map}\;h\;ss)$ for all $a \in A$ and $as \in A^{\infty}$;
(5) $h\,(\mathit{iLm}^{\mathcal{A}}\;xs\;a) = \mathit{iLm}^{\mathcal{B}}\;xs\;(h\;a)$ for all $xs \in \mathit{iVar}^{\infty,\neq}$ and $a \in A$.

Prop 57. $\mathcal{ILT}\mathit{erm} = (\mathit{ILTerm}, \_[\_], \mathit{FV}, \mathit{iVr}, \mathit{iLm}, \mathit{iAp})$ is initial in the category of iλ-enriched QLS-nominal sets. More explicitly, for any iλ-enriched QLS-nominal set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, \mathit{Supp}^{\mathcal{A}}, \mathit{iVr}^{\mathcal{A}}, \mathit{iLm}^{\mathcal{A}}, \mathit{iAp}^{\mathcal{A}})$, there exists a unique morphism from $\mathcal{ILT}\mathit{erm}$ to $\mathcal{A}$, i.e., a function $h : \mathit{ILTerm} \to A$ satisfying the following properties:

(1) $h\,(t[\sigma]) = (h\;t)[\sigma]^{\mathcal{A}}$ for all $\sigma \in \mathit{Perm}$ and $t \in \mathit{ILTerm}$;
(2) $\mathit{Supp}^{\mathcal{A}}\;(h\;t) \subseteq \mathit{FV}\;t$ for all $t \in \mathit{ILTerm}$;
(3) $h\,(\mathit{iVr}\;x) = \mathit{Vr}^{\mathcal{A}}\;x$ for all $x \in \mathit{iVar}$;
(4) $h\,(\mathit{iAp}\;t\;ts) = \mathit{iAp}^{\mathcal{A}}\;(h\;t)\;(\mathit{map}\;h\;ts)$ for all $t \in \mathit{ILTerm}$ and $ts \in \mathit{ILTerm}^{\infty}$;
(5) $h\,(\mathit{iLm}\;xs\;t) = \mathit{iLm}^{\mathcal{A}}\;xs\;(h\;t)$ for all $xs \in \mathit{iVar}^{\infty,\neq}$ and $t \in \mathit{ILTerm}$.

D.3 π-calculus processes as an abstract datatype

In this subsection, *Var* is a countable set of variables (a.k.a. names, or channels), and *Perm* is the set of permutations on *Var*, here meaning bijections of finite support. In §8.1 we omitted from the grammar some π-calculus syntax constructors. We give the full grammar here:

$$P \quad ::= \quad 0 \;\mid\; P + Q \;\mid\; P \parallel Q \;\mid\; !P \;\mid\; [x = y]P \;\mid\; [x \neq y]P \;\mid\; \overline{a}\,x.\,P \;\mid\; a(x).\,P \;\mid\; \nu(x).\,P$$

So the π-calculus processes, forming the set *Proc*, are generated by the constructors:

- $0 \in \mathit{Proc}$
- $+ \;: \mathit{Proc} \to \mathit{Proc} \to \mathit{Proc}$
- $\parallel \;: \mathit{Proc} \to \mathit{Proc} \to \mathit{Proc}$
- $[\_ = \_]\,\_ : \mathit{Var} \to \mathit{Var} \to \mathit{Proc} \to \mathit{Proc}$
- $[\_ \neq \_]\,\_ \;: \mathit{Var} \to \mathit{Var} \to \mathit{Proc} \to \mathit{Proc}$
- $\overline{\_}\,\_.\,\_ \;: \mathit{Var} \to \mathit{Var} \to \mathit{Proc} \to \mathit{Proc}$ (output)
- $\_(\_).\,\_ \;: \mathit{Var} \to \mathit{Var} \to \mathit{Proc} \to \mathit{Proc}$ (input)
- $\nu(\_).\,\_ \;: \mathit{Var} \to \mathit{Proc} \to \mathit{Proc}$

Of these constructors, all are free except for the last two (which introduce bindings). In addition to the constructors, we also have the free-variable operator $\mathit{FV} : \mathit{Proc} \to \mathcal{P}_{\mathsf{fin}}(\mathit{Var})$ and permutation operator $\_[\_] : \mathit{Proc} \to \mathit{Perm} \to \mathit{Proc}$.

Lemma 58. (Distinctness and (quasi-)injectivity of the constructors) The following hold:
(1) The processes $0,\; P + Q,\; P_1 \parallel Q_1,\; !P_2,\; [x = y]P_3,\; [x_1 \neq y_1]P_4,\; \overline{a}\,x_2.\,P,\; a_1(x_3).\,P,\; \nu(x_4).\,P_5$ are all distinct;
(2) $P + Q = P' + Q'$ iff $P = P'$ and $Q = Q'$;
(3) $P \parallel Q = P' \parallel Q'$ iff $P = P'$ and $Q = Q'$;
(4) $!P = !P'$ iff $P = P'$;
(5) $[x = y]P = [x' = y']P'$ iff $x = x'$, $y = y'$ and $P = P'$;
(6) $[x \neq y]P = [x' \neq y']P'$ iff $x = x'$, $y = y'$ and $P = P'$;
(7) $\overline{a}\,x.\,P = \overline{a'}\,x'.\,P'$ iff $a = a'$, $x = x'$ and $P = P'$;
(8) $a(x).\,P = a'(x').\,P'$ iff $a = a'$ and there exists y that is fresh for a, a', x, x', P, P' (i.e., $y \notin \{a, a', x, x'\} \cup \mathit{FV}\;P \cup \mathit{FV}\;P'$) such that $P[y \leftrightarrow x] = P'[y \leftrightarrow x']$;
(9) $\nu(x).\,P = \nu(x').\,P'$ iff there exists y that is fresh for x, x', P, P' (i.e., $y \notin \{x, x'\} \cup \mathit{FV}\;P \cup \mathit{FV}\;P'$) such that $P[y \leftrightarrow x] = P'[y \leftrightarrow x']$.

Lemma 59. (Equivariance of the constructors) The following hold, assuming $\sigma \in Bij_{<\aleph_0}(Var)$:
(1) $0[\sigma] = 0$;
(2) $(P + Q)[\sigma] = P[\sigma] + Q[\sigma]$;
(3) $(P \parallel Q)[\sigma] = P[\sigma] \parallel Q[\sigma]$;
(4) $(!P)[\sigma] = !\,(P[\sigma])$;
(5) $([x = y]P)[\sigma] = [\sigma\ x = \sigma\ y]\ (P[\sigma])$;
(6) $([x \neq y]P)[\sigma] = [\sigma\ x \neq \sigma\ y]\ (P[\sigma])$;
(7) $(\overline{a}\,x.\,P)[\sigma] = \overline{\sigma\ a}\,\sigma\ x.\,P[\sigma]$;
(8) $(a(x).\,P)[\sigma] = \sigma\ a\ (\sigma\ x).\,P[\sigma]$;
(9) $(\nu(x).\,P)[\sigma] = \nu(\sigma\ x).\,P[\sigma]$.

Lemma 60. (Free variables versus constructors) The following hold:
(1) $FV\ 0 = \emptyset$;
(2) $FV\ (P + Q) = FV\ P \cup FV\ Q$;
(3) $FV\ (P \parallel Q) = FV\ P \cup FV\ Q$;
(4) $FV\ (!P) = FV\ P$;
(5) $FV\ ([x = y]P) = FV\ P \cup \{x, y\}$;
(6) $FV\ ([x \neq y]P) = FV\ P \cup \{x, y\}$;
(7) $FV\ (\overline{a}\,x.\,P) = FV\ P \cup \{a, x\}$;
(8) $FV\ (a(x).\,P) = (FV\ P \setminus \{x\}) \cup \{a\}$;
(9) $FV\ (\nu(x).\,P) = FV\ P \setminus \{x\}$.

Lemma 61. (Structural induction) Assume $\varphi : Proc \to Bool$ is a predicate such that the following hold:

- $\varphi\ 0$;
- $\forall P, Q.\ \varphi\ P \land \varphi\ Q \longrightarrow \varphi\ (P + Q)$;
- $\forall P, Q.\ \varphi\ P \land \varphi\ Q \longrightarrow \varphi\ (P \parallel Q)$;
- $\forall P.\ \varphi\ P \longrightarrow \varphi\ (!P)$;
- $\forall x, y, P.\ \varphi\ P \longrightarrow \varphi\ ([x = y]P)$;
- $\forall x, y, P.\ \varphi\ P \longrightarrow \varphi\ ([x \neq y]P)$;
- $\forall a, x, P.\ \varphi\ P \longrightarrow \varphi\ (\overline{a}\,x.\,P)$;
- $\forall a, x, P.\ \varphi\ P \longrightarrow \varphi\ (a(x).\,P)$;
- $\forall x, P.\ \varphi\ P \longrightarrow \varphi\ (\nu(x).\,P)$.

Then $\forall P.\ \varphi\ P$.

Lemma 62. (Strong structural induction, can be obtained from plain structural induction using Thm. 7) Assume $(P, Psupp : P \to \mathcal{P}_{\mathrm{fin}}(Var))$ is a parameter structure and $\varphi : P \to Proc \to Bool$ is a predicate such that the following hold:

- $\forall p.\ \varphi\ p\ 0$;
- $\forall p, P, Q.\ (\forall q.\ \varphi\ q\ P) \land (\forall q.\ \varphi\ q\ Q) \longrightarrow \varphi\ (P + Q)$;
- $\forall p, P, Q.\ (\forall q.\ \varphi\ q\ P) \land (\forall q.\ \varphi\ q\ Q) \longrightarrow \varphi\ (P \parallel Q)$;
- $\forall p, P.\ (\forall q.\ \varphi\ q\ P) \longrightarrow \varphi\ (!P)$;
- $\forall p, x, y, P.\ (\forall q.\ \varphi\ q\ P) \longrightarrow \varphi\ p\ ([x = y]P)$;
- $\forall p, x, y, P.\ (\forall q.\ \varphi\ q\ P) \longrightarrow \varphi\ p\ ([x \neq y]P)$;
- $\forall p, a, x, P.\ (\forall q.\ \varphi\ q\ P) \longrightarrow \varphi\ p\ (\overline{a}\,x.\,P)$;
- $\forall p, a, x, P.\ x \notin Psupp\ p \cup \{a\} \land (\forall q.\ \varphi\ q\ P) \longrightarrow \varphi\ p\ (a(x).\,P)$;
- $\forall p, x, P.\ x \notin Psupp\ p \land (\forall q.\ \varphi\ q\ P) \longrightarrow \varphi\ p\ (\nu(x).\,P)$.

Then $\forall p, P.\ \varphi\ p\ P$.

Note that above, in the last but one hypothesis (for the input case), we allow ourselves to assume the freshness of x not only for the parameter p, but also for the binding-“passive” part of the process, a.

Next we describe the corresponding instance of the recursor from [Blanchette et al. 2019]. For brevity, we will skip some intermediate concepts, e.g., morphisms, and describe more directly the end product.

Def 63. A *π-enriched QLS-nominal set* is a tuple $\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, Supp^{\mathcal{B}}, SZero, Plus^{\mathcal{B}}, Par^{\mathcal{B}}, Bang^{\mathcal{B}}, Match^{\mathcal{B}}, Mismatch^{\mathcal{B}}, Out^{\mathcal{B}}, Inp^{\mathcal{B}}, Nu^{\mathcal{B}})$ such that $(B, \_[\_]^{\mathcal{B}}, Supp^{\mathcal{B}})$ is a QLS-nominal set, and

- $SZero \in B$
- $Plus^{\mathcal{B}} : B \to B \to B$
- $Par^{\mathcal{B}} : B \to B \to B$
- $Bang^{\mathcal{B}} : B \to B$
- $Match^{\mathcal{B}} : Var \to Var \to B \to B$
- $Mismatch^{\mathcal{B}} : Var \to Var \to B \to B$
- $Out^{\mathcal{B}} : Var \to Var \to B \to B$
- $Inp^{\mathcal{B}} : Var \to Var \to B \to B$
- $Nu^{\mathcal{B}} : Var \to B \to B$

are operators, such that the following hold:

- all the operators are equivariant;
- $Supp^{\mathcal{B}}\ SZero = \emptyset$;
- $Supp^{\mathcal{B}}\ (Plus^{\mathcal{B}}\ b_1\ b_2) \subseteq Supp^{\mathcal{B}}\ b_1 \cup Supp^{\mathcal{B}}\ b_2$;
- $Supp^{\mathcal{B}}\ (Par^{\mathcal{B}}\ b_1\ b_2) \subseteq Supp^{\mathcal{B}}\ b_1 \cup Supp^{\mathcal{B}}\ b_2$;
- $Supp^{\mathcal{B}}\ (Bang^{\mathcal{B}}\ b) \subseteq Supp^{\mathcal{B}}\ b$;
- $Supp^{\mathcal{B}}\ (Match^{\mathcal{B}}\ x\ y\ b) \subseteq Supp^{\mathcal{B}}\ b \cup \{x, y\}$;
- $Supp^{\mathcal{B}}\ (Mismatch^{\mathcal{B}}\ x\ y\ b) \subseteq Supp^{\mathcal{B}}\ b \cup \{x, y\}$;
- $Supp^{\mathcal{B}}\ (Out^{\mathcal{B}}\ a\ x\ b) \subseteq Supp^{\mathcal{B}}\ b \cup \{a, x\}$;
- $Supp^{\mathcal{B}}\ (Inp^{\mathcal{B}}\ a\ x\ b) \subseteq (Supp^{\mathcal{B}}\ b \setminus \{x\}) \cup \{a\}$;
- $Supp^{\mathcal{B}}\ (Nu^{\mathcal{B}}\ x\ b) \subseteq Supp^{\mathcal{B}}\ b \setminus \{x\}$.

Prop 64. $\mathcal{P}roc = (Proc,\ \_[\_],\ FV,\ 0,\ +,\ \parallel,\ [\_ = \_]\_,\ [\_ \neq \_]\_,\ \bar{\_}\_,\ \_(\_),\ \nu(\_).\_)$ is the initial π-enriched QLS-nominal sets. More explicitly, for any π-enriched QLS-nominal set $\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, Supp^{\mathcal{B}}, SZero, SPlus, SPar, SBang, SMatch, SMismatch, SOut, SInp, SNu)$, there exists a unique morphism from $\mathcal{P}roc$ to $\mathcal{B}$, i.e., a function $h : Proc \to B$ satisfying the following properties:

(1) $h\,(t[\sigma]) = (h\ t)[\sigma]^{\mathcal{B}}$ (assuming $\sigma \in Perm$);
(2) $Supp^{\mathcal{A}}\ (h\ t) \subseteq FV\ t$;
(3) $h\ 0 = Zero^{\mathcal{B}}$;
(4) $h\,(P + Q) = Plus^{\mathcal{B}}\ (h\ P)\ (h\ Q)$;
(5) $h\,(P \parallel Q) = Par^{\mathcal{B}}\ (h\ P)\ (h\ Q)$;
(6) $h\,(!P) = Bang^{\mathcal{B}}\ (h\ P)$;
(7) $h\,([x = y]P) = Match^{\mathcal{B}}\ x\ y\ (h\ P)$;
(8) $h\,([x \neq y]P) = Mismatch^{\mathcal{B}}\ x\ y\ (h\ P)$;
(9) $h\,(\overline{a}\,x.\,P) = Out^{\mathcal{B}}\ a\ x\ (h\ P)$;
(10) $h\,(a(x).\,P) = Inp^{\mathcal{B}}\ a\ x\ (h\ P)$;
(11) $h\,(\nu(x).\,P) = Nu^{\mathcal{B}}\ x\ (h\ P)$.

D.4 The syntax of infinitary FOL, $\mathcal{L}_{\kappa_1,\kappa_2}$, as an abstract datatype

In this subsection, we consider the setting from §9.1: κ_1, κ_2 are infinite cardinals, $\kappa = \max(\kappa_1, \kappa_2)$, *Var* is a set of variables of cardinality κ, and $Perm = Perm_\kappa$ is the set of permutations on *Var*, meaning here bijections of κ-small support.

Recall from §9.1 that the $\mathcal{L}_{\kappa_1,\kappa_2}$-formulas, forming the set $Fmla = Fmla_{\kappa_1,\kappa_2}$, are generated by the constructors $Eq : Var \to Var \to Fmla$, $Neg : Fmla \to Fmla$, $Conj : \mathcal{P}_{<\kappa_1}(Fmla) \to Fmla$, and $All : \mathcal{P}_{<\kappa_2}(Var) \to Fmla \to Fmla$. Of these constructors, *Eq*, *Neg* and *Conj* are free, and *All* is not. In addition to the constructors, we also have the free-variable operator $FV : Fmla \to \mathcal{P}_{<\kappa}(Var)$ and permutation operator $\_[\_] : Fmla \to Perm \to Fmla$.

Among all our example datatypes in this paper, this one that recursive through a "permutative" type constructor (here, that of $\mathcal{P}_{<\kappa_1}$) and is also the first one to bind sets of variables. The latter will be reflected in the slightly more elaborate quasi-injectivity property for the binding constructor *All*, Lemma 65(5): Since we no longer have fixed positions for the variables in the binders, we must control their correspondence via explicit permutations.

Lemma 65. (Distinctness and (quasi-)injectivity of the constructors) The following hold:
(1) The formulas $Eq\ x\ y$, $Neg\ f$, $Conj\ F$ and $All\ V\ f$ are all distinct;
(2) $Eq\ x\ y = Eq\ x'\ y'$ iff $x = x'$ and $y = y'$;
(3) $Neg\ f = Neg\ f'$ iff $f = f'$;
(4) $Conj\ f F = Conj\ F'$ iff $F = F'$;
(5) $All\ V\ f = All\ V'\ f'$ iff there exists $\sigma, \sigma' \in Perm$ such that
- $Im\ \sigma\ V = Im\ \sigma'\ V'$,
- $Im\ \sigma\ V$ is fresh for V, V', f, f' (i.e., $Im\ \sigma\ V \cap (V \cup V' \cup FV\ f \cup FV\ f') = \emptyset$), and
- $f[\sigma] = f'[\sigma']$.

An alternative (simpler but asymmetric) formulation of the quasi-injectivity of *All* (point (5) above) is the following:
(5') $All\ V\ f = All\ V'\ f'$ iff there exists $\sigma \in Perm$ such that
- $Im\ \sigma\ V = V'$,
- V is fresh for $All\ V'\ f'$ (i.e., $V \cap FV\ (All\ V'\ f') = \emptyset$), and
- $f[\sigma] = f'$.

Lemma 66. (Equivariance of the constructors) The following hold, assuming $\sigma \in Perm$:
(1) $(Eq\ x\ y)[\sigma] = Eq\,(\sigma\ x)\,(\sigma\ y)$;
(2) $(Neg\ f)[\sigma] = Neg\,(f[\sigma])$;
(3) $(Conj\ F)[\sigma] = Conj\,(Im\ (\_[\sigma])\ F)$;
(4) $(All\ V\ f)[\sigma] = All\,(Im\ \sigma\ V)\,(f[\sigma])$.

Lemma 67. (Free variables versus constructors) The following hold:
(1) $FV\ (Eq\ x\ y) = \{x, y\}$;
(2) $FV\ (Neg\ f) = FV\ f$;
(3) $FV\ (Conj\ F) = \bigcup\ (Im\ FV\ F)$;
(4) $FV\ (All\ V\ f) = FV\ f \setminus V$.

Lemma 68. (Structural induction) Assume $\varphi : Fmla \to Bool$ is a predicate such that the following hold:

- $\forall x, y.\ \varphi\ (Eq\ x\ y)$;
- $\forall f.\ \varphi\ f \longrightarrow \varphi\ (Neg\ f)$;
- $\forall F.\ (\forall f \in F.\ \varphi\ f) \longrightarrow \varphi\ (Conj\ F)$;
- $\forall V, f.\ \varphi\ f \longrightarrow \varphi\ (All\ V\ f)$.

Then $\forall t.\ \varphi\ t$.

Lemma 69. (Strong structural induction, can be obtained from plain structural induction using Thm. 20) Assume $(P, Psupp : P \to \mathcal{P}_{<\kappa}(iVar))$ is a parameter structure and $\varphi : P \to Fmla \to Bool$ is a predicate such that the following hold:

- $\forall p, x, y.\ \varphi\ p\ (Eq\ x\ y)$;
- $\forall p, f.\ (\forall q.\ \varphi\ q\ f) \longrightarrow \varphi\ p\ (Neg\ f)$;
- $\forall p, F.\ (\forall f \in F.\ (\forall q.\ \varphi\ q\ f)) \longrightarrow \varphi\ p\ (Conj\ F)$;
- $\forall p, V, f.\ V \cap Psupp\ p = \emptyset \wedge (\forall q.\ \varphi\ q\ f) \longrightarrow \varphi\ (All\ V\ f)$.

Then $\forall p, t.\ \varphi\ p\ t$.

Finally, we describe the corresponding instance of the recursor from [Blanchette et al. 2019].

Def 70. An $\mathcal{L}_{\kappa_1,\kappa_2}$-*enriched QLS-nominal set* is a tuple $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, Eq^{\mathcal{A}}, Neg^{\mathcal{A}}, Conj^{\mathcal{A}}, All^{\mathcal{A}})$ such that $(A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ is a QLS-nominal set and $Eq^{\mathcal{A}} : Var \to Var \to A$, $Neg^{\mathcal{A}} : A \to A$, $Conj^{\mathcal{A}} : \mathcal{P}_{<\kappa_1}(A) \to A$, and $All : \mathcal{P}_{<\kappa_2}(Var) \to A \to A$ are operators, such that the following hold:

- $Eq^{\mathcal{A}}$, $Neg^{\mathcal{A}}$, $Conj^{\mathcal{A}}$ and $All^{\mathcal{A}}$ are equivariant;
- $Supp^{\mathcal{A}}\ (Eq^{\mathcal{A}}\ x\ y) \subseteq \{x, y\}$;
- $Supp^{\mathcal{A}}\ (Neg^{\mathcal{A}}\ f) \subseteq Supp^{\mathcal{A}}\ f$;
- $Supp^{\mathcal{A}}\ (Conj^{\mathcal{A}}\ F) \subseteq \bigcup\ (Im\ Supp^{\mathcal{A}}\ F)$;
- $Supp^{\mathcal{A}}\ (All^{\mathcal{A}}\ V\ f) \subseteq Supp^{\mathcal{A}}\ f \setminus V$.

Prop 71. $\mathcal{F}mla = (Fmla, \_[\_], FV, Eq, Neg, Conj, All)$ is initial in the category of $\mathcal{L}_{\kappa_1,\kappa_2}$-enriched QLS-nominal sets. More explicitly, for any $\mathcal{L}_{\kappa_1,\kappa_2}$-enriched QLS-nominal set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, Eq^{\mathcal{A}}, Neg^{\mathcal{A}}, Conj^{\mathcal{A}}, All^{\mathcal{A}})$, there exists a unique morphism from $\mathcal{F}mla$ to $\mathcal{A}$, i.e., a function $h : Fmla \to A$ satisfying the following properties:

(1) $h\,(f[\sigma]) = (h\ f)[\sigma]^{\mathcal{A}}$ for all $\sigma \in Perm$ and $f \in Fmla$;
(2) $Supp^{\mathcal{A}}\ (h\ f) \subseteq FV\ f$ for all $f \in Fmla$;
(3) $h\,(Eq\ x\ y) = Eq^{\mathcal{A}}\ x\ y$ for all $x \in iVar$;
(4) $h\,(Neg\ f) = Neg^{\mathcal{A}}\ (h\ f)$ for all $f \in Fmla$;
(5) $h\,(Conj\ F) = Conj^{\mathcal{A}}\ (Im\ h\ F)$ for all $F \in \mathcal{P}_{<\kappa_1}(Fmla)$.
(6) $h\,(All\ V\ f) = All^{\mathcal{A}}\ V\ (h\ f)$ for all $f \in Fmla$ and $V \in \mathcal{P}_{<\kappa_2}(Var)$.

E MORE DETAILS ON THE INFINITARY LAMBDA CALCULUS CASE STUDY

Here we give details on the isomorphism between the (finitary) λ-calculus and the uniform affine infiniary λ-calculus established by Mazza [2012], which we have mechanized in Isabelle taking advantage of our developed binder-aware datatype, recursion and induction infrastructure. We will highlight the places where binding-aware reasoning is essential. We will use the concepts and notations from §9.3. The presentation will avoid any Isabelle jargon—we defer to §G the discussion of Isabelle-specific aspects.

We start by recalling the following nuance: The finitary λ-calculus is defined over a countable set of variables, i.e., of cardinality $\aleph_0$, whereas the infinitary one is defined over an uncountable set of variables, namely of cardinality $\aleph_1$. (This nuance is not addressed by Mazza, who works with the same countable set of variables for both calculi; more about this in §E.9.) We therefore write *Var* for the countable set of variables used by finitary λ-calculus, and *iVar* for the uncountable set of variables used by the infinitary λ-calculus, and refer to the elements of *iVar* as *ivariables*.

$$iAp\ (iLm\ xs\ t)\ ts \Rightarrow t[ts/xs]\ \text{(iBeta)} \qquad \frac{t \Rightarrow t'}{iLm\ xs\ t \Rightarrow iLm\ xs\ t'}\ \text{(iXi)}$$

$$\frac{t \Rightarrow t'}{iAp\ t\ ts \Rightarrow iAp\ t'\ ts}\ \text{(iApL)} \qquad \frac{i \in \mathbb{N} \qquad ts_i \Rightarrow t'}{iAp\ t\ ts \Rightarrow iAp\ t\ (ts[i := t'])}\ \text{(iApR)}$$

Fig. 16. Plain β-reduction for iterms

E.1 Plain infinitary β-reduction

As a warm-up, in Fig. 16 we define the straightforward notion of β-reduction on iterms, $\Rightarrow$: $ILTerm \to ILTerm \to Bool$. Thus, similarly to β-reduction for the fininary λ-calculus, we have the reduction of the β-redexes (rule (iBeta)), which can take place under any sequence of abstractions and applications (rules (iXi), (iApL) and (iApR)). The differences from the finitary case come from the very structure of the syntax: Since now we bind not individual variables but entire (nonrepetitive) streams of variables, β-reduction substitutes all these variables simultaneously; moreover, in the right rule for application, we choose one position i in the stream ts that constitutes the application's second argument. (We write $(ts[i := t'])$ for the stream obtained from ts by replacing, on its position i, ts_i with t'.)

The strong rule induction for this relation is obtained by instantiating our (equivariance-based) strong induction criterion (Thm. 7):

Prop 72. Let $(P, Psupp : P \to \mathcal{P}_{\text{countable}}(Var))$ be a parameter structure. Let $\varphi : P \to ILTerm \to ILTerm \to Bool$ and assume the following hold:

- (iBeta) case: $\forall p, xs, t, ts.$
  $set\ xs \cap (Psupp\ p \cup \bigcup_{t' \in set\ ts} FV\ t') = \emptyset$
  $\longrightarrow \varphi\ p\ (iAp\ (iLm\ xs\ t)\ ts)\ (t[ts/xs])$
- (iXi) case: $\forall p, xs, t, t'.$
  $set\ xs \cap Psupp\ p = \emptyset \wedge (t \Rightarrow t') \wedge (\forall q.\ \varphi\ q\ t\ t')$
  $\longrightarrow \varphi\ p\ (iLm\ xs\ t)\ (iLm\ xs\ t')$
- (iApL) case: $\forall p, t, t', ts.$
  $(t \Rightarrow t') \wedge (\forall q.\ \varphi\ q\ t\ t')$
  $\longrightarrow \varphi\ p\ (iAp\ t\ ts)\ (iAp\ t'\ ts)$
- (iApR) case: $\forall p, t, ts, i, t'.$
  $(ts_i \Rightarrow t') \wedge (\forall q.\ \varphi\ q\ ts_i\ t')$
  $\longrightarrow \varphi\ p\ (iAp\ t\ ts')\ (iAp\ t\ (ts'[i := t']))$

Then $\forall p, t, t'.\ (t \Rightarrow t') \longrightarrow \varphi\ p\ t\ t'$.

Note that, in the (iBeta) case, the obtained strong induction principle allows us to avoid the variables not only of the parameter p, but also of the "passive" terms of the rule, namely all the terms in ts. By contrast, prior state of the art (provided it would have been extended to apply to infinitary syntax), rather than *offering* freshness of x for ts as a bonus of the exported induction, would instead amend the statement of the (iBeta) rule from Fig. 16 to *require* this freshness condition in the first place.

This "formal bonus" for strong induction can bring some convenience in proofs—for example, when proving that the *affine* predicate (defined in §9.3) is preserved by β-reduction:

Lemma 73. If *affine* t and $t \Rightarrow t'$ then *affine* t'.

The proof of this must go by rule induction on $t \Rightarrow t'$. Here, in the (iBeta) case, standard rule induction would require us to show that $\mathit{affine}\ (\mathit{iAp}\ (\mathit{iLm}\ \mathit{xs}\ t)\ \mathit{ts})$ implies $\mathit{affine}\ (t[\mathit{ts}/\mathit{xs}])$, whereas strong induction (Prop. 21) with empty set of parameters would additionally allow us to assume that $\mathit{set}\ \mathit{xs} \cap \bigcup_{t' \in \mathit{set}\ \mathit{ts}} \mathit{FV}\ t' = \emptyset$. Both alternatives require an additional lemma expressing that *affine* is preserved by substitution. But with the strong induction alternative the following lemma, naturally assuming affinity and disjointness conditions between the involved terms, suffices:

Lemma 74. If $\mathit{affine}\ t$, $(\forall i, j.\ \mathit{FV}\ \mathit{ts}_i \cap \mathit{FV}\ \mathit{ts}_j = \emptyset)$ and $(\forall i.\ \mathit{affine}\ \mathit{ts}_i \wedge \mathit{FV}\ t \cap \mathit{FV}\ \mathit{ts}_i = \emptyset)$, then $\mathit{affine}\ (t[\mathit{ts}/\mathit{xs}])$.

By contrast, with the standard induction alternative, we would need a stronger and more subtle (and harder to prove) version of the substitution lemma:

Lemma 75. If $\mathit{affine}\ t$, $(\forall i, j.\ \mathit{FV}\ \mathit{ts}_i \cap \mathit{FV}\ \mathit{ts}_j = \emptyset)$ and $(\forall i.\ \mathit{affine}\ \mathit{ts}_i \wedge \mathit{FV}\ t \cap \mathit{FV}\ \mathit{ts}_i \subseteq \mathit{set}\ \mathit{xs})$, then $\mathit{affine}\ (t[\mathit{ts}/\mathit{xs}])$.

In turn, the proof of either of the above two lemmas requires strong rule induction for the *affine* predicate (Prop. 21), where, as usual, the freshness assumptions allow substitution to be pushed inside abstractions.

While β-reduction as defined above preserves affinineness, it does not preserve the notion of uniformity required for the isomorphism with the finitary calculus, so Mazza introduces a different one. But before formalizing that, we need some properties of renaming equivalence (the relation underlying uniformity).

E.2 More on renaming equivalence

Recall the renaming equivalence relation $\approx\ : \mathit{ILTerm} \to \mathit{ILTerm} \to \mathit{Bool}$ defined in §9.3 and the strong rule induction associated to it, Prop. 22. When introducing this relation, Mazza briefly notes that it is symmetric and transitive (but not reflexive).

Lemma 76. The following hold for all $t, t', t'' \in \mathit{ILTerm}$:
(1) $t \approx t'$ implies $t' \approx t$.
(2) $t \approx t'$ and $t' \approx t''$ implies $t \approx t''$.

While symmetry (point (1)) follows routinely by standard rule induction, proving transitivity (point (2)) requires some work. In a proof by rule induction on $t \approx t'$, in the inductive case for abstractions (iLm), we know (as inductive hypothesis) that $\forall t''.\ t \approx t' \wedge t' \approx t'' \longrightarrow t \approx t''$, and we also know $\mathit{iLm}\ \mathit{xs}\ t \approx \mathit{iLm}\ \mathit{xs}\ t' \approx s''$ where $\mathit{xs} \in \mathit{Super}$, and must show $\mathit{iLm}\ \mathit{xs}\ t \approx s''$. For this, we must be able to show that s'' has the form $\mathit{iLm}\ \mathit{xs}\ t''$ for some t'' such that $t' \approx t''$, which would allow us to apply the inductive hypothesis to obtain $t \approx t''$, and then apply the (iLm) rule to prove what we wanted. So we need the following inversion lemma for $\approx$ w.r.t. abstractions:[6]

Lemma 77. If $\mathit{xs} \in \mathit{Super}$ and $\mathit{iLm}\ \mathit{xs}\ t \approx s'$, then there exists t' such that $s' = \mathit{iLm}\ \mathit{xs}\ t'$ and $t \approx t'$.

This last lemma is proved as follows: From the standard inversion rule for $\approx$ and the distinctness of the iterm constructors, we obtain $\mathit{ys} \in \mathit{Super}$ and s_1, t_1 such that (1) $t_1 \approx s_1$, (2) $\mathit{iLm}\ \mathit{xs}\ t = \mathit{iLm}\ \mathit{ys}\ t_1$ and (3) $s' = \mathit{iLm}\ \mathit{ys}\ s_1$. We take t' to be $s_1[(\mathit{map}\ \mathit{iVr}\ \mathit{xs})/\mathit{ys}]$, where *map* is the mapping function for streams. Now, $s' = \mathit{iLm}\ \mathit{xs}\ t'$ follows from (3) and the properties of abstraction and substitution. Moreover, from (2) and the properties of abstraction and substitution, we obtain that

[6]We also need corresponding lemmas w.r.t. variable injections and applications, but these are straightforward to prove thanks to the injectiveness of *iVr* and *iAp*.

$t = t_1[(map\ iVr\ xs)/ys]$. So $t \approx t'$ amounts to $t_1[(map\ iVr\ xs)/ys] \approx s_1[(map\ iVr\ xs)/ys]$, which can be inferred from (1) and the following:

Lemma 78. If $xs, ys \in Super$ and $t \approx t'$ then $t[(map\ iVr\ xs)/ys] \approx t'[(map\ iVr\ xs)/ys]$.

The last lemma follows from a more general result, stating that (under certain conditions), $\approx$ preserves substitution. But before stating that we need to take stock of another property of $\approx$ (which follows by standard induction), namely that any two renaming-equivalent iterms touch the same supervariables, and touch only finitely many of them:

Lemma 79. If $t \approx t'$, then the sets $\{xs \in Super \mid set\ xs \cap FV\ t \neq \emptyset\}$ and $\{xs \in Super \mid set\ xs \cap FV\ t' \neq \emptyset\}$ are equal, and finite.

Now the mentioned more general result:

Lemma 80. (Lemma 11 from [Mazza 2012]) If $t \approx t'$, $xs \in Super$ and $(\forall t_1, t_2.\ \{t_1, t_2\} \subseteq set\ ts \cup set\ ts' \longrightarrow t_1 \approx t_2)$, then $t[ts/xs] \approx t'[ts'/xs]$.

The proof of this lemma goes by strong rule induction on $t \approx t'$ (Prop. 22). We take the parameters to be triples (xs, ts, ts'), and $Psupp\,(xs, ts, ts') = set\ xs \cup \bigcup_{t \in set\ ts \cup set\ ts'} FV\ t$. Each $Psupp\,(xs, ts, ts')$ is obviously countable, and also touches a finite number of supervariables because:

- $set\ xs$ touches only the supervariable xs;
- all iterms in $set\ ts \cup set\ ts'$ being mutually renaming equivalent, by Lemma 79 they touch exactly the finite set of supervariables that some ts_i does (for some i).

Thanks to being able to assume, in the (iLm) case, that the binding stream of variables is fresh for xs, ts and ts', the substitutions $\_[ts/xs]$ and $\_[ts'/xs]$ can be pushed inside the abstractions (as easy as they inside applications) and the proof goes smoothly.

This concludes the journey of proving that $\approx$ transitive, which on the way also gathered some reusable lemmas, including an inversion and a substitutivity lemma for $\approx$; strong rule induction was required for the latter. (Interestingly, while Mazza concludes that $\approx$ is transitive immediately after defining this relation and only later states his Lemma 11 (our Lemma 80), our formal analysis reveals the usefulness of proving Lemma 11 *before* transitivity, in order to help in the transitivity proof.)

E.3 Uniformity and uniform infinitary β-reduction

Renaming equivalence is symmetric and transitive but not reflexive, i.e., it is a partial equivalence relation (PER). An iterm t is said to be *uniform*, written *uniform* t, provided it is renaming-equivalent to itself, $t \approx t$. Let us call a stream of iterms ts uniform, written $uniformS\ ts$, if $\forall i, j.\ ts_i \approx ts_j$. In particular, in a uniform stream ts each ts_i is uniform, but the condition is much stronger than that—as any two iterms ts_i and ts_j are required to be renaming equivalent, in particular, have the same (iLm, iAp, iLm)-structure as trees.

So uniformity of an iterm means that all injected variables belong to some supervariables, all binders in abstractions are supervariables, and all iterm streams from the righthand side of applications are uniform (via *uniformS*).

For later usage, let us note the following inversion rule for uniformity of abstractions, as an immediate consequence of Lemma 77:

Lemma 81. If $xs \in Super$ and $uniform\,(iLm\ xs\ t)$ then $uniform\ t$.

From the previous discussion it should be clear that β-reduction as defined in Appendix E.1 appe does not preserve uniformity. This is mostly because it allows reducing a single iterm in the righthand side stream of iterms of an application. Mazza addresses this by introducing a different

$$\frac{\mathit{uniformS}\ ts \qquad \forall i.\ ts_i \Rightarrow_{\mathsf{head}} ts'_i}{ts \Rightarrow_0 ts'}\ (\text{iBeta}) \qquad \frac{xs \in \mathit{Super} \qquad ts \Rightarrow_k ts'}{\mathit{map}\ (\mathit{iLm}\ xs)\ ts\ \Rightarrow_k\ \mathit{map}\ (\mathit{iLm}\ xs)\ ts'}\ (\text{iXi})$$

$$\frac{\mathit{uniformSS}\ tss \qquad ts \Rightarrow_k ts'}{\mathit{map}_2\ \mathit{iAp}\ ts\ tss\ \Rightarrow_{k+1}\ \mathit{map}_2\ \mathit{iAp}\ ts'\ tss}\ (\text{iApL}) \qquad \frac{\mathit{uniformS}\ ts \qquad \mathit{flat}\ tss \Rightarrow_k \mathit{flat}\ tss'}{\mathit{map}_2\ \mathit{iAp}\ ts\ tss\ \Rightarrow_{k+1}\ \mathit{map}_2\ \mathit{iAp}\ ts\ tss'}\ (\text{iApR})$$

Fig. 17. Uniform β-reduction for streams of iterms

notion, which we call *uniform (infinitary) β-reduction*. It is a ternary relation $\Rightarrow : \mathit{ILTerm} \to \mathbb{N} \to \mathit{ILTerm} \to \mathit{Bool}$, where we write $t \Rightarrow_k t'$ for its application to the iterms t, t' and the number k. The numeric argument simply tracks the applicative depth of the redexes, i.e., the number of applications under which reduction occurs; it is meant to offer a more precise description of reduction, and is orthogonal to the notion of uniformity. Mazza's inductive definition of this relation [Mazza 2012, Def. 7] is extremely informal, much more so than the rest of his definitions—in the inductive case for righthand side of application, he writes the following (where we paraphrase to use our notations):

> "if ts is such that $\forall i, j.\ ts_i \approx ts_j$ and $ts_0 \Rightarrow_k t'_0$, by uniformity the 'same' reduction can be performed in all ts_i, obtaining the term t'_i. If we define ts' such that $ts'_i = t'_i$ for all i, we set $\mathit{iAp}\ t\ ts \Rightarrow_{k+1}\ \mathit{iAp}\ t\ ts'$."

To make this rigorous, we must inductively describe a form of parallel reduction of a *stream of iterms*, making sure that the same redex is reduced in all members of the stream. To this end, we define β-reduction not on iterms, but on streams on iterms, $\Rightarrow : \mathit{ILTerm}^\infty \to \mathbb{N} \to \mathit{ILTerm}^\infty \to \mathit{Bool}$. The definition, shown in Fig. 17, uses several auxiliary operators.

- In the (iBeta) rule, we use the head-reduction relation $\Rightarrow_{\mathsf{head}} : \mathit{ILTerm} \to \mathit{ILTerm} \to \mathit{Bool}$ defined as follows: $t_1 \Rightarrow_{\mathsf{head}} t_2$ iff there exist xs, t and ts such that $t_1 = \mathit{iAp}\ (\mathit{iLm}\ xs\ t)\ ts$ and $t_2 = t[ts/xs]$. (Thus t_1 becomes t_2 by reducing a redex located at its "head", i.e., top.)
- In the (iApL) rule, we use the predicate *uniformSS*, which is the further extension of *uniform* and *uniformS* to streams of streams (i.e., stream matrices) of iterms: $\mathit{uniformSS}\ tss = (\forall i, i', j, j'.\ tss_{i,i'} \approx tss_{j,j'})$; and map_2 (binary stream-map), applied to any function $f : U \to V \to W$ (such as *iAp*) and two streams $us \in U^\infty$ and $vs \in V^\infty$, yields the stream obtained by applying f componentwise to these: $(\mathit{map}_2\ f\ us\ vs)_i = f\ us_i\ vs_i$ for all i.
- In the (iApR) rule, $\mathit{flat}\ tss$ is the flattening (via "dovetailing") of the stream of streams of iterms tss into a single stream of iterms; formally, we have a bijection $b : \mathbb{N}^2 \to \mathbb{N}$ between the pairs of indexes of tss and the indexes of $\mathit{flat}\ tss$ such that the elements correspond to each other, in that $(\mathit{flat}\ tss)_{b(i,j)} = tss_{i,j}$.

A few notes on the design decisions for this definition:

- Mazza intends his reduction relation to work on uniform terms only. In our formalization, we acheive this by adding uniformity conditions only when necessary, namely on the source of head-reduction in (iBeta) and on the reduction-passive terms in (iApL) and (iApR); the uniformity of all the other involved terms does not need to be stated, as it follows inductively from the definition. Indeed, one of the sanity checks we prove by standard rule induction is the following:

  Lemma 82. (corresponds to Prop. 14(1) from [Mazza 2012]) If $ts \Rightarrow_k ts'$, then $\mathit{uniformS}\ ts$ and $\mathit{uniformS}\ ts'$.

Proving this requires the following lemma when dealing with the (iBeta) case, which clarifies why we required
uniformS ts but not *uniformS ts′* in the (iBeta) rule:

Lemma 83. If $ts \Rightarrow_{\mathsf{head}} ts'$ and *uniformS ts*,
then *uniformS ts′*.

- The use of the combinator *flat* is essential in the (iApR) rule. Indeed, for example replacing the $\mathit{flat}\ tss \Rightarrow_k \mathit{flat}\ tss'$ hypothesis with something like $\mathit{lift}_2\ (\Rightarrow_k)\ tss\ tss'$ (meaning $\forall i.\ tss_i \Rightarrow_k tss'_i$), thus lifting $\Rightarrow_k$ componentwise to streams of streams, would not achieve the desired result. This is because we want all the iterms in the "matrix" *tss* to reduce by exactly the "same reduction", which is achieved by the flattening approach—whereas the lifting approach would only achieve "sameness" inside each column of the matrix, not across different columns.

We obtain the strong rule induction for uniform β-reduction by instantiating our relative-equivariance criterion, Thm. 23, with parameters similar to the ones we used for the renaming equivalence strong induction (described in §9.3 and the end of §9.4). This is no surprise, since the rules for uniform reduction use (derivatives of) the *uniform* predicate, which in turn is defined from renaming equivalence.

Prop 84. Let $(P, \mathit{Psupp} : P \to \mathcal{P}_{\mathsf{countable}}(\mathit{Var}))$ be a parameter structure such that, for any $p \in P$, $\{xs \in \mathit{Super} \mid \mathit{set}\ xs \cap \mathit{Psupp}\ p \neq \emptyset\}$ is finite. Let $\varphi : P \to \mathit{ILTerm} \to \mathbb{N} \to \mathit{ILTerm} \to \mathit{Bool}$ and assume the following hold:

- (iBeta) case: $\forall p, ts, ts'.\ \mathit{uniformS}\ ts \wedge (\forall i.\ ts_i \Rightarrow_{\mathsf{head}} ts'_i)$
  $\longrightarrow \varphi\ p\ ts\ 0\ ts'$
- (iXi) case: $\forall p, xs, ts, k, ts'.$
  $\mathit{set}\ xs \cap \mathit{Psupp}\ p = \emptyset \wedge (ts \Rightarrow_k ts') \wedge (\forall q.\ \varphi\ q\ ts\ k\ ts')$
  $\longrightarrow \varphi\ p\ (\mathit{map}\ (\mathit{iLm}\ xs)\ ts)\ k\ (\mathit{map}\ (\mathit{iLm}\ xs)\ ts')$
- (iApL) case: $\forall p, ts, k, ts', tss.\ \mathit{uniformSS}\ tss \wedge$
  $(ts \Rightarrow_k ts') \wedge (\forall q.\ \varphi\ q\ ts\ k\ ts')$
  $\longrightarrow \varphi\ p\ (\mathit{map}\ (\mathit{iAp}\ ts)\ tss)\ (k+1)\ (\mathit{map}\ (\mathit{iAp}\ ts')\ tss)$
- (iApR) case: $\forall p, ts, tss, k, tss'.\ \mathit{uniformS}\ ts \wedge$
  $(\mathit{flat}\ tss \Rightarrow_k \mathit{flat}\ tss') \wedge (\forall q.\ \varphi\ q\ (\mathit{flat}\ tss)\ k\ (\mathit{flat}\ tss'))$
  $\longrightarrow \varphi\ p\ (\mathit{map}\ (\mathit{iAp}\ ts)\ tss)\ (k+1)\ (\mathit{map}\ (\mathit{iAp}\ ts)\ tss')$

Then $\forall p, ts, k, ts'.\ (ts \Rightarrow_k ts') \longrightarrow \varphi\ p\ ts\ k\ ts'$.

Note that, unlike in the case of the other β-reduction relations we discussed so far (e.g., the finitary β-reduction from Fig. 1 and plain infinitary β-reduction from Fig. 16), here the associated strong rule induction allows the parameter freshness assumption only for the inductive λ-case (here, (iXi)) and not for the reduction base case (here (iBeta)). This is because this time in the base case we have "hidden" the involved binding structure inside a different relation, $\Rightarrow_{\mathsf{head}}$, and the strong rule induction for an inductively defined relation does not cross the boundaries of its auxiliary relations such as $\Rightarrow_{\mathsf{head}}$; nor we believe it should, for the sake of modularity. Instead, for such auxiliary relations we can take care of the desired freshness enhancement separately, for example we can prove the following lemma for $\Rightarrow_{\mathsf{head}}$ (where we highlight what this lemma brings in addition to the definition of $\Rightarrow_{\mathsf{head}}$):

Lemma 85. If a countable set of variables A is such that $\{xs \in \mathit{Super} \mid \mathit{set}\ xs \cap A \neq \emptyset\}$ is finite, and if t_1 is uniform and $t_1 \Rightarrow_{\mathsf{head}} t_2$, then there exist $xs \in \mathit{Super}$, t and ts such that $t_1 = \mathit{iAp}\ (\mathit{iLm}\ xs\ t)\ ts$ and $t_2 = t[ts/xs]$ and $\mathit{set}\ xs \cap A = \emptyset$.

So using Prop. 84 in conjunction with Lemma 85 (taking A to be $Psupp\ p$) enables fresh assumptions for both (iBeta) and (iXi). It should be noted that for the other β-reduction relations we could have also hidden the binding structure in the base case under an auxiliary head reduction relation, to the same effect on the generated strong rule induction; and solutions similar to Lemma 85 could have been used to address that. This shows that the outreach of strong rule induction is less essential for axioms than for proper induction rules, since for the former we can easily deploy alternative ad hoc solutions. In particular, going back to our motivating examples in §2, the strong rule induction benefits are less essential for axioms such as (Beta) than for rules such as (ParBeta) and (Xi).

E.4 Translating finitary to infinitary λ-terms

Remember that *Super* denotes the countable set of supervariables, consisting of nonrepetitive streams xs of ivariables such that any two distinct streams $xs, ys \in Super$ are disjoint, in that $set\ xs \cap set\ ys = \emptyset$.

Let $superOf : Var \to Super$ be a fixed bijection between the (countable) sets of variables and supervariables; we will write $superOf^{-1} : Super \to Var$ for its inverse.

We will call *position* any element p of $\mathbb{N}^*$, i.e., any word (list) over natural numbers, and let $natOf : \mathbb{N}^* \to \mathbb{N}$ be a fixed injection between positions and natural numbers. Given $p \in \mathbb{N}^*$ and $n \in \mathbb{N}$, we will write $p \cdot n$ for the word obtained from p by adding n at the end.

The set *Super* and the functions *superOf* and *natOf* will all be parameters of the to-be-defined translation; their exact choice does not matter beyond having to satisfy their above stated properties.

Mazza defines his finitary-to-infinitary translation as a function $[\![\_]\!]_\_ : LTerm \to \mathbb{N}^* \to ILTerm$, recursively by the following equations:

(3) $[\![Vr\ x]\!]_p = iVr\ ((superOf\ x)_{natOf\ p})$
(4) $[\![Lm\ x\ t]\!]_p = iLm\ (superOf\ x)\ [\![t]\!]_p$
(5) $[\![Ap\ t_1\ t_2]\!]_p = iAp\ [\![t_1]\!]_{p\cdot 0}\ ([\![t_2]\!]_{p\cdot 1}, [\![t_2]\!]_{p\cdot 2}, [\![t_2]\!]_{p\cdot 3}, \ldots)$

(A slightly more succinct way to write the righthand side of the equation for application is

$$iAp\ [\![t_1]\!]_{p\cdot 0}\ (map\ [\![t_2]\!]_{p\cdot \_}\ (natsFrom\ 1))$$

where, for any n, $natsFrom\ n$ denotes the stream of natural numebrs starting from n, namely $[n, n+1, n+2, \ldots]$.)

The intuition is that every variable x in the original term is duplicated in the translation into countably many ivariable "copies" of it sourced from its corresponding supervariable, $superOf\ x$. The positions are used to make sure that the copies located inside different parts of the resulted iterm are distinct, thus ensuring that the iterm is affine. Indeed, in the recursive case for application, we see that the position p grows with different numbers postpended on the different arguments of infinitary application, which ensures disjointness in conjunction with choosing the particular "copy" based on this position counter ($natOf\ p$) when reaching the Vr-leaves. Correspondingly, abstraction over a variable is translated to abstraction over its supervariable, i.e., over all its "copies".

Since terms are alpha-equivalence classes, in particular the constructor Lm is not injective, equations (3)–(5) above are not *a priori* guaranteed to form a correct definition. To make them into a rigorous definition, we deploy Prop. 49's recursion principle. This requires us to organize the target domain $A = (\mathbb{N}^* \to ILTerm)$ into a λ-enriched QLS-nominal set. The constructor-like operators on A, namely $Vr^{\mathcal{A}}$, $Ap^{\mathcal{A}}$ and $Lm^{\mathcal{A}}$, are determined by the above recursive equations; for example, for any $x \in Var$ and $a \in A$, we take $Lm^{\mathcal{A}}\ x\ a$ to be $\lambda p.\ iLm\ (superOf\ x)\ (a\ p)$. As for the permutation and support operators, we determine them by formulating answers to the questions on how should the to-be-defined function $[\![\_]\!]_\_$ interact with permutation and free variables, namely, for $t \in LTerm$ and $\sigma \in Perm$,

- $[\![t[\sigma]]\!]_p = ?$
- $? \subseteq FV\ t$

where the question marks must be replaced with expressions depending on $[\![t]\!]_\_$, i.e., on the application of $[\![\_]\!]_\_$ to t (and to positions possibly different from p).

Note that answering these questions is likely to be useful anyway (and it will!) for later proofs involving this translation—just that the recursor requires us to think these through in advance. Upon analysis, the following possible answers emerge:

(1) $[\![t[\sigma]]\!]_p = [\![t]\!]_p[v2iv\,\sigma]$
(2) $Im\ superOf^{-1}\ (TouchedSuper\ [\![t]\!]_p) \subseteq FV\ t$

where:

- $v2iv\,\sigma$ (read "variable to ivariable") is the conversion of $\sigma : Var \to Var$, via $superOf$, into a supervariable-structure preserving function on $iVar \to iVar$; namely, for any $y \in iVar$ such that y appears in some (necessarily unique) supervariable xs, we define $v2iv\,\sigma\ y$ as $(superOf\ (\sigma\ (superOf^{-1}\ xs)))_i$ for the unique i such that $xs_i = y$.
- For any $t \in ILTerm$, $TouchedSuper\ t$ is the set of all supervariables that are touched by (the free variables of) s, namely $\{xs \in Super \mid set\ xs \cap FV\ t \neq \emptyset\}$

Equation (1) above is seen to be quite intuitive if we remember that the translation sends variables to supervariables, which means that bijections σ between variables naturally correspond to bijections between supervariables, hence (thanks to the supervariables being mutually disjoint) to supervariable-structure preserving bijections between ivariables; therefore indeed (A) applying a bijection on variables and then translating should be the same as (B) first translating and then applying this corresponding bijection of its ivariable "copies" in the translation.

As for the above inclusion (2), we obtained it by adjunction from

$$TouchedSuper\ [\![t]\!]_p \subseteq Im\ superOf\ (FV\ t),$$

which is again intuitive if we think in terms of the variable-supervariable correspondence.

(A good approach for coming up with (1) and (2) is to "pretend" that that $[\![\_]\!]_\_$ has already been defined via (3)–(5) and think about formulating lemmas describing its behavior w.r.t. mapping and free-variables.)

With the structure on $(\mathbb{N}^* \to ILTerm)$ determined by Mazza's recursive clauses (3)–(5) together with clauses (1) and (2), we would like to check that $(\mathbb{N}^* \to ILTerm)$ becomes a λ-enriched QLS-nominal set. However, this is not true while working with the entire set $ILTerm$. Among other things that go wrong, the $TouchedSuper$ operator does not behave well on iterms that are non-uniform. The solution comes from remembering that the translation is aimed to target not arbitrary, but uniform iterms. So restricting the target domain to the subset $K \subseteq (\mathbb{N}^* \to ILTerm)$ consisting of mutually renaming-equivalent (in particular uniform) position interpretations only, namely $K = \{u : \mathbb{N}^* \to ILTerm \mid \forall p, p'.\ u\ p \approx u\ p'\}$, does the job. Indeed, it is now routine to check that K becomes a λ-enriched QLS-nominal set, and therefore Prop. 49 legitimates Mazza's definition as well as our two additional (tentative) properties, obtaining:

Lemma 86. There exists a unique function $[\![\_]\!]_\_ : K \to ILTerm$ such that the above clauses (1)–(5) hold.

Immediately from the definition of K, we have the following:

Lemma 87. (Lemma 15(2) from [Mazza 2012]) For all $t \in LTerm$ and $p, p' \in \mathbb{N}^*$, we have $[\![t]\!]_p \approx [\![t]\!]_{p'}$; in particular, $[\![t]\!]_p$ is uniform.

By routine structural induction on $t \in LTerm$ we can also prove that all the free variables of $[\![t]\!]_p$ that appear in some supervariable must necessarily appear there at a position higher than p (w.r.t. the prefix order, via *natOf*):

Lemma 88. For all $t \in LTerm$, $p, p' \in \mathbb{N}^*$ and $xs \in Super$, if $xs_{natOf\ p'} \in FV\ [\![t]\!]_p$ then p is a prefix of p'.

This immediately implies:

Lemma 89. (statement inlined in Mazza's proof sketch for Lemma 15(1) in [Mazza 2012]) For all $t \in LTerm$ and $p, p' \in \mathbb{N}^*$, if p and p' are incomparable w.r.t. the prefix order then $FV\ [\![t]\!]_p \cap FV\ [\![t]\!]_{p'} = \emptyset$.

Using the above, the affinity of all iterms in the image of the translation follows routinely by structural induction:

Lemma 90. (Lemma 15(1) from [Mazza 2012]) For all $t \in LTerm$ and $p \in \mathbb{N}^*$, we have that $[\![t]\!]_p$ is affine.

E.5 Translating infinitary to finitary terms

For the translation $(\!|\_|\!)$ in the opposite direction, i.e., from infinitary (back to) finitary terms, Mazza writes the following equations:

(3) $(\!| iVr\ xs_i |\!) = Vr\ (superOf^{-1}\ xs)$
(4) $(\!| iLm\ xs\ t |\!) = Lm\ (superOf^{-1}\ xs)\ (\!| t |\!)$
(5) $(\!| iAp\ t\ ts |\!) = Ap\ (\!| t |\!)\ (\!| ts_0 |\!)$

These recursive equations are clearly intended not for arbitrary iterms, but for uniform ones:

- The bound stream of variables xs from $iLm\ xs\ t$ in equation (4) is assumed to be a supervariable, since the inverse of the *superOf* function is being applied to it.
- In the application case, equation (5), all the terms in ts but the first one (ts_0) are ignored, which is only meaningful if no essential information is lost—as guaranteed when the iterm $iAp\ t\ ts$ is uniform, making all the iterms ts_i mutually renaming equivalent.

And indeed, Mazza explicitly restricts his definition to uniform iterms, writing the type of $(\!|\_|\!)$ as $\{t \in ILTerm \mid uniform\ t\} \rightarrow LTerm$ (again paraphrasing to use our notations).

While it is possible to extend equations (3)–(5) above to an attempted definition on the entire set of iterms (not just uniform ones), e.g., performing an arbitrary choice when xs in $iLm\ xs\ t$ is not a supervariable, this would make it hard to deploy the nominal-style recursor for the syntax of infinitary λ-calculus expressed in Prop. 57 (as well as, it seems, any potential infinitary generalization of other nominal-style recursors, which are very close to each other in terms of expressiveness [Popescu 2024]). And this is no surprise, given the above observation that $(\!|\_|\!)$ is not intended to work on the *entire* set of iterms, which is what Prop. 57's recursor specializes in.

E.6 A custom, supervariable-sensitive recursor

So instead, we develop a custom recursor specialized in a subdomain of iterms that are "good" (well-behaved) w.r.t. the supervariable infrastructure.[7] Namely, we define the predicate $good : ILTerm \rightarrow Bool$ inductively as in Fig. 18.

We chose the predicate *good* to be a sweet spot between uniformity (which is unary but not inductive hence not recursion-friendly) and renaming equivalence (which is inductive but binary,

[7]The ideas are likely generalizable to a recursor on restricted domains of terms with bindings, subject to some abstract conditions; but we have not yet investigated such a potential generalization.

$$\frac{xs \in Super \qquad x \in set\ xs}{good\ (iVr\ x)}\ (\text{iVr}) \qquad\qquad \frac{xs \in Super \qquad good\ t}{good\ (iLm\ xs\ t)}\ (\text{iLm})$$

$$\frac{good\ t \qquad \forall t' \in set\ ts.\ good\ t' \qquad \forall t_1, t_2.\ \{t_1, t_2\} \subseteq set\ ts \longrightarrow TouchedSuper\ t_1 = TouchedSuper\ t_2}{good\ (iAp\ t\ ts)}\ (\text{iAp})$$

Fig. 18. The *good* predicate on iterms

hence in itself not suitable as a domain-restricting predicate). Like renaming equivalence, it requires the injected variables to belong to some supervariables (in rule (iVr)) and the bound streams of variables to be supervariables (in rule (iLm)). Moreover, in rule (iAp) we require a property that we know it holds for renaming equivalence and uniformity (thanks to Lemma 79), namely that all iterms from the righthand side stream of terms in applications touch exactly the same supervariables. This is a way to ensure that good terms touch only finitely many supervariables; indeed, this follows by standard rule induction:

Lemma 91. *good* t implies that *TouchedSuper* t is finite for all $t \in ILTerm$.

Moreover, that goodness is a sound approximation of renaming equivalence (hence also of uniformity) can be proved by standard rule induction, using Lemma 79:

Lemma 92. $t \approx t'$ implies *good* t and *good* t' for all $t, t' \in ILTerm$. In particular, *uniform* t implies *good* t for all $t \in ILTerm$.

Thm. 23 also applies to this predicate, yielding a strong rule induction principle similar to those for renaming equivalence and uniform β-reduction.

Prop 93. Let $(P, Psupp : P \to \mathcal{P}_{\text{countable}}(Var))$ be a parameter structure such that, for any $p \in P$, $\{xs \in Super \mid set\ xs \cap Psupp\ p \neq \emptyset\}$ is finite. Let $\varphi : P \to ILTerm \to Bool$ and assume the following hold:

- (iVr) case: $\forall p, xs, x.\ xs \in Super \wedge x \in set\ xs \longrightarrow \varphi\ p\ (iVr\ x)$
- (iLm) case: $\forall p, xs, t. set\ xs\ \cap\ Psupp\ p = \emptyset \wedge xs \in Super \wedge good\ t \wedge (\forall q.\ \varphi\ q\ t)\ \longrightarrow\ \varphi\ p\ (iLm\ xs\ t)$
- (iAp) case: $\forall p, t, ts.\ good\ t\ \wedge\ (\forall q.\ \varphi\ q\ t)\ \wedge\ (\forall t' \in set\ ts.\ good\ t'\ \wedge\ (\forall q.\ \varphi\ q\ t'))\ \wedge$
  $(\forall t_1, t_2.\ \{t_1, t_2\} \subseteq set\ ts \longrightarrow TouchedSuper\ t_1 = TouchedSuper\ t_2)$
  $\longrightarrow\ \varphi\ p\ (iAp\ t\ ts)$

Then $\forall p, t.\ good\ t \longrightarrow \varphi\ p\ t$.

In what follows we formulate a recursion principle for defining functions on good iterms, so in particular one that is sensitive to the notion of supervariable. To this end, we introduce variations of the notions of permutative and QLS-nominal sets that take supervariables into account; we focus on the case of the set of variables being *iVar*, hence its cardinal κ being $\aleph_1$.

Def 94. For any set of ivariables $A \subseteq iVar$, we let *TSuper* A be the set of its touched supervariables, namely $\{xs \in Super \mid set\ xs\ \cap\ A \neq \emptyset\}$. (Note that, for any $t \in ILTerm$, we have $TouchedSuper\ t = TSuper\ (FV\ t)$.)

A (countable-core) permutation σ is called *Super-sensitive* when:

- it is *Super*-compatible, in that it preserves the supervariables (via mapping): for all $xs \in Var^{\infty,\neq}$, if $xs \in Super$ then $map\ \sigma\ xs \in Super$;
- its core (i.e., support) touches only finitely many supervariables, in that $TSuper\ (Core\ \sigma)$ is finite.

We let $Perm_{Super}$ denote the set of *Super*-sensitive (countable-core) permutations.

A *Super-sensitive pre-nominal set* is a pair $\mathcal{A} = (A, \_[\_]^{\mathcal{A}})$ where A and $\_[\_] : A \to Perm_{Super} \to A$ is an action of $Perm_{Super}$ on A.

A *Super-sensitive QLS-nominal set* is a triple $\mathcal{A} = (A, \_[\_], Supp^{\mathcal{A}})$ where:

- $(A, \_[\_]^{\mathcal{A}})$ is a *Super*-sensitive pre-nominal set;
- $Supp^{\mathcal{A}} : S \to \mathcal{P}_{\text{countable}}(Var)$ is such that $(\forall xs \in TSuper(Supp^{\mathcal{A}}\ a).\ map\ \sigma; xs = xs)$ implies $a[\sigma]^{\mathcal{A}} = a$ for all $\sigma \in Perm_{Super}$ and $a \in A$.

Now we define a corresponding variation of the notion of iλ-enriched QLS-nominal set:

Def 95. A *Super-sensitive iλ-enriched QLS-nominal set* is a tuple $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, iVr^{\mathcal{A}}, iLm^{\mathcal{A}}, iAp^{\mathcal{A}})$ such that $(A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}})$ is a *Super*-sensitive QLS-nominal set, and $iVr^{\mathcal{A}} : iVar \to A$, $iAp^{\mathcal{A}} : A \to A^{\infty} \to A$ and $iLm^{\mathcal{A}} : iVar^{\infty,\neq} \to A \to A$ are operators such that the following hold:

- $iVr^{\mathcal{A}}$, $iAp^{\mathcal{A}}$ and $iLm^{\mathcal{A}}$ are equivariant w.r.t. *Super*-sensitive permutations;
- $TSuper\,(Supp^{\mathcal{A}}\,(iVr^{\mathcal{A}}\ x)) \subseteq TSuper\,\{x\}$ for all $x \in iVar$;
- $TSuper\,(Supp^{\mathcal{A}}\,(iAp^{\mathcal{A}}\ a\ as)) \subseteq$
  $TSuper\,(Supp^{\mathcal{A}}\ a) \cup \bigcup_{a' \in set\ as} TSuper\,(Supp^{\mathcal{A}}\ a')$
  for all $a \in A$ and $as \in A^{\infty}$;
- $TSuper\,(Supp^{\mathcal{A}}\,(iLm^{\mathcal{A}}\ xs\ a)) \subseteq TSuper\,(Supp^{\mathcal{A}}\ a) \setminus \{xs\}$ for all $xs \in Super$ and $a \in A$.

The difference between this variation and the original (Def. 55) is that everything is conditioned by supervariable sensitivity or membership to some supervariable, and the inclusions between the sets of variables are not "raw" as before but mediated through the sets of touched supervariables. So, roughly speaking, we have a relativization of the original concept w.r.t. supervariables. And the same goes for morphisms:

Def 96. Given two *Super-sensitive iλ-enriched QLS-nominal sets* $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, iVr^{\mathcal{A}}, iLm^{\mathcal{A}}, iAp^{\mathcal{A}})$ and $\mathcal{B} = (B, \_[\_]^{\mathcal{B}}, Supp^{\mathcal{B}}, iVr^{\mathcal{B}}, iLm^{\mathcal{B}}, iAp^{\mathcal{B}})$, a morphism between $\mathcal{A}$ and $\mathcal{B}$ is a function $h : A \to B$ that commutes or sub-commutes with the operators, in the following sense:

(1) $h\,(a[\sigma]^{\mathcal{A}}) = (h\ a)[\sigma]^{\mathcal{B}}$ for all $\sigma \in Perm_{Super}$ and $a \in A$;
(2) $TSuper\,(Supp^{\mathcal{B}}\,(h\ a)) \subseteq TSuper\,(Supp^{\mathcal{A}}\ a)$ for all $a \in A$;
(3) $h\,(iVr^{\mathcal{A}}\ x) = iVr^{\mathcal{B}}\ x$ for all $xs \in Super$ and $x \in iVar$ such that $x \in set\ xs$;
(4) $h\,(iAp^{\mathcal{A}}\ a\ as) = iAp^{\mathcal{B}}\ (h\ s)\ (map\ h\ as)$ for all $s \in A$ and $as \in A^{\infty}$ such that $\forall a_1, a_2.\ \{a_1, a_2\} \subseteq set\ as \longrightarrow TSuper\,(Supp^{\mathcal{A}}\ a_1) = TSuper\,(Supp^{\mathcal{A}}\ a_2)$;
(5) $h\,(iLm^{\mathcal{A}}\ xs\ a) = iLm^{\mathcal{B}}\ xs\ (h\ a)$ for all $xs \in Super$ and $a \in A$.

Note that, in the clause (4) above, we condition the commutation of h with the application operators by the arguments having the same touched supervariables, similarly to what we did when defining the *good* predicate on iterms.

It is easy to see that the set $\{t \in ILTerm \mid good\ t\}$ is closed under the term constructors and mapping, and that $\mathcal{ILTerm}_{Super} = (\{t \in ILTerm \mid good\ t\}, \_[\_], FV, iVr, iLm, iAp)$ is a *Super*-sensitive iλ-enriched QLS-nominal set. In fact, we can show that it is the initial one, which gives us a recursion principle for good terms:

Prop 97. $\mathcal{ILTerm}_{Super}$ is initial in the category of *Super*-sensitive iλ-enriched QLS-nominal sets. More explicitly, for any iλ-enriched QLS-nominal set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, iVr^{\mathcal{A}}, iLm^{\mathcal{A}}, iAp^{\mathcal{A}})$, there exists a unique morphism from $\mathcal{ILTerm}_{Super}$ to $\mathcal{A}$, i.e., a function $h : \{t \in ILTerm \mid good\ t\} \to A$ satisfying the following properties:

(1) $h\,(t[\sigma]) = (h\ a)[\sigma]^{\mathcal{A}}$ for all $\sigma \in Perm_{Super}$ and $t \in ILTerm$ such that $good\ t$;
(2) $TSuper\,(Supp^{\mathcal{A}}\,(h\ t)) \subseteq TouchedSuper\ t$ for all $t \in ILTerm$ such that $good\ t$;

(3) $h\,(iVr\ x) = SVr\ x$ for all $xs \in Super$ and $x \in iVar$ such that $x \in set\ xs$;
(4) $h\,(iAp\ t\ ts) = iAp^{\mathcal{A}}\ (h\ t)\ (map\ h\ ts)$ for all $t \in ILTerm$ and $ts \in ILTerm^{\infty}$ such that $good\ t$, $\forall t' \in ts.\ good\ t'$ and $\forall t_1, t_2.\ \{t_1, t_2\} \subseteq set\ ts \longrightarrow TouchedSuper\ t_1 = TouchedSuper\ t_2$;
(5) $h\,(iLm\ xs\ t) = iLm^{\mathcal{A}}\ xs\ (h\ t)$ for all $xs \in Super$ and $t \in ILTerm$ such that $good\ t$.

E.7 Back to translating infinitary to finitary terms

We can now deploy Prop. 97's recursor to complete (and make rigorous) the definition given by Mazza's clauses (3)–(5) from §E.5, which we rephrase here taking goodness into account:

(3) $(\!|iVr\ x|\!) = Vr\,(superOf^{-1}\ xs)$ for all $xs \in Super$ and $x \in set\ xs$.
(4) $(\!|iLm\ xs\ t|\!) = Lm\ (superOf^{-1}\ xs)\ (\!|t|\!)$ for all $xs \in Super$ and $t \in ILTerm$ such that $good\ t$.
(5) $(\!|iAp\ t\ ts|\!) = Ap\ (\!|t|\!)\ (\!|ts_0|\!)$ for all $t \in ILTerm$ and $ts \in ILTerm^{\infty}$ such that $good\ t$, $(\forall t' \in ts.\ good\ t')$ and $(\forall t_1, t_2.\ \{t_1, t_2\} \subseteq set\ ts \longrightarrow TouchedSuper\ t_1 = TouchedSuper\ t_2)$.

Note that (5) can equivalently be written as:

(5) $(\!|iAp\ t\ ts|\!) = Ap\ (\!|t|\!)\ (\!|ts_0|\!)$ for all $t \in ILTerm$ and $ts \in ILTerm^{\infty}$ such that $good\ (iAp\ t\ ts)$.

Using Prop. 97, we will turn the above into a recursive definition of a function $(\!|\_|\!) : \{t \in ILTerm \mid good\ t\} \longrightarrow A$, where $A = LTerm$. To this end, we will organize A as a $Super$-sensitive iλ-enriched QLS-nominal set $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, iVr^{\mathcal{A}}, iLm^{\mathcal{A}}, iAp^{\mathcal{A}})$. Similarly to how we proceeded to deploying the (finitary) λ-term recursor in §E.4, we have that $iVr^{\mathcal{A}}$, $iLm^{\mathcal{A}}$ and $iAp^{\mathcal{A}}$ are determined by the above clauses; and we determine the permutation and support operators by analyzing how the to-be-defined function $(\!|\_|\!)$ should interact with mapping and free variables, namely:

- $(\!|t[\sigma]|\!) = ?$ for all $t \in ILTerm$ and $\sigma \in Perm_{Super}$ such that $good\ t$,
- $? \subseteq TouchedSuper\,(FV\ t)$ for all $t \in ILTerm$ such that $good\ t$,

where the question marks must be replaced with expressions depending on $(\!|t|\!)$. Here again, we can give some natural answers to these questions:

(1) $(\!|t[\sigma]|\!) = (\!|t|\!)[iv2v\,\sigma]$ for all $t \in ILTerm$ and $\sigma \in Perm_{Super}$ such that $good\ t$.
(2) $Im\ superOf\ (FV\ (\!|t|\!)) \subseteq TouchedSuper\,(FV\ t)$ for all $t \in ILTerm$ such that $good\ t$.

Above, $iv2v$ (read "ivariable to variable") is the inverse of the conversion operator $v2iv$ used in the opposite translation (from finitary to infinitary terms, in §E.4). It converts, via $superOf$, a $Super$-sensitive function $g : iVar \rightarrow iVar$ into a function $iv2v\,g : Var \rightarrow Var$. It is defined by $iv2v\,g = \lambda x.\ superOf^{-1}\ (map\ g\ (superOf\ x))$.

With the structure determined by clauses (1)–(5), it is routine to check that $\mathcal{A} = (A, \_[\_]^{\mathcal{A}}, Supp^{\mathcal{A}}, iVr^{\mathcal{A}}, iLm^{\mathcal{A}}, iAp^{\mathcal{A}})$ is an iλ-enriched QLS-nominal set, which, via Prop. 97, gives us:

Lemma 98. There exists a unique function $[\![\_]\!] : \{t \in ILTerm \mid good\ t\} \longrightarrow S$ such that the above clauses (1)–(5) hold.

Note that Mazza's informal definition of $[\![\_]\!]$ has a different type, namely $[\![\_]\!] : \{t \in ILTerm \mid uniform\ t\} \longrightarrow S$. But because $uniform$ implies $good$ (by Lemma 92), the restriction of our defined function to uniform terms gives us Mazza's exact version (plus the clauses (1) and (2) as "bonus"):

Lemma 99. There exists a unique function $[\![\_]\!] : \{t \in ILTerm \mid uniform\ t\} \longrightarrow S$ such that the following clauses hold:

(1) $(\!|t[\sigma]|\!) = (\!|t|\!)[iv2v\,\sigma]$ for all $t \in ILTerm$ and $\sigma \in Perm_{Super}$ such that $uniform\ t$.
(2) $Im\ superOf\ (FV\ (\!|t|\!)) \subseteq TouchedSuper\,(FV\ t)$ for all $t \in ILTerm$ such that $uniform\ t$.
(3) $(\!|iVr\ x|\!) = Vr\,(superOf^{-1}\ xs)$ for all $xs \in Super$ and $x \in set\ xs$.
(4) $(\!|iLm\ xs\ t|\!) = Lm\ (superOf^{-1}\ xs)\ (\!|t|\!)$ for all $xs \in Super$ and $t \in ILTerm$ such that $uniform\ t$.
(5) $(\!|iAp\ t\ ts|\!) = Ap\ (\!|t|\!)\ (\!|ts_0|\!)$ for all $t \in ILTerm$ and $ts \in ILTerm^{\infty}$ such that $uniform\ (iAp\ t\ ts)$.

$$Ap\ (Lm\ x\ t_1)\ t_2\ \rightarrow_0\ t_1[t_2/x]\ \text{(Beta)} \qquad \frac{t \rightarrow_k t'}{Lm\ x\ t\ \rightarrow_k\ Lm\ x\ t'}\ \text{(Xi)}$$

$$\frac{t_1 \rightarrow_k t_1'}{Ap\ t_1\ t_2\ \rightarrow_{k+1}\ Ap\ t_1'\ t_2}\ \text{(ApL)} \qquad \frac{t_2 \rightarrow_k t_2'}{Ap\ t_1\ t_2\ \rightarrow_{k+1}\ Ap\ t_1\ t_2'}\ \text{(ApR)}$$

Fig. 19. λ-calculus β-reduction indexed by the applicative depth of the redex

So we can in principle regard *good* as an auxiliary over-approximation of *uniform* that helped us complete the formal definition of $(\!|\_|\!)$. However, retaining the more general type $\{t \in ILTerm \mid good\ t\} \rightarrow S$ for $(\!|\_|\!)$ will be helpful beyond this goal, as it will allow us to do proofs by rule induction on *good*.

E.8 The isomorhism

To summarize what we have so far:

- Each term $s \in LTerm$ is translated, for each position $p \in \mathbb{N}^*$, to an iterm $[\![s]\!]_p \in ILTerm$ that is both uniform and affine; and in fact, for all $p, q \in \mathbb{N}^*$, $[\![s]\!]_p$ and $[\![s]\!]_q$ are renaming equivalent.
- Each good, in particular, each uniform iterm $t \in ILTerm$ is translated to a term $(\!|t|\!)$.

Mazza's goal is to show that the two translations give an isomorphism between (1) λ-terms under β-reduction and (2) equivalence classes of uniform affine items modulo renaming-equivalence[8] under (infinitary) uniform β-reduction. (Recall that we already know from Lemma 82 that uniform β-reduction ensures the uniformity of its participating terms.)

To avoid lack of clarity, we will use single arrow for β-reduction on (finitary) terms, and keep using double arrow for uniform β-reduction on iterms. In fact, to synchronize the two and state Mazza's result faithfully, we introduce the indexed version of the former, tracking the applicative depth of the redex like we did for the latter. Thus, we define $\rightarrow : LTerm \rightarrow \mathbb{N} \rightarrow LTerm$ as in Fig. 19. We omit showing the generated strong induction principle, since it is very similar to that of plain β-reduction.

Mazza's main result consists of a sequence of five statements, which in our formalization looks as follows:

Thm 100. The following hold:
(1) (Lemma 16 from [Mazza 2012]) $t \approx t'$ implies $(\!|t|\!) = (\!|t'|\!)$ for all $t, t' \in ILTerm$.
(2) (Thm. 19(1) from [Mazza 2012]) $(\!|[\![s]\!]_p|\!) = s$ for all $s \in LTerm$ and $p \in \mathbb{N}^*$.
(3) (Thm. 19(2) from [Mazza 2012]) $[\![(\!|t|\!)]\!]_p \approx t$ for all $t \in ILTerm$ and $p \in \mathbb{N}^*$ such that *uniform* t.
(4) (corresponds to Thm. 19(3) from [Mazza 2012]) For all $s, s' \in LTerm$, $k \in \mathbb{N}$ and $ps \in (\mathbb{N}^*)^\infty$, if $s \rightarrow_k s'$ then there exists ts' such that $(map\ [\![s]\!]\_\ ps) \Rightarrow_k ts'$ and $lift_2\ (\approx)\ ts'\ (map\ [\![s']\!]\_\ ps)$ (in particular, *uniformS* ts').
(5) (corresponds to Thm. 19(4) from [Mazza 2012]) For all $ts, ts' \in ILTerm^\infty$ and $k \in \mathbb{N}$, if $ts \Rightarrow_k ts'$ then $lift_2\ (\rightarrow_k)\ (map\ (\!|\_|\!)\ ts)\ (map\ (\!|\_|\!)\ ts')$.

(Recall that, for any two streams as, as' over some set A and binary relation R on A, we write $lift_2\ R\ as\ as'$ for the componenwise lifting of the relation to these streams, namely $\forall i.\ R\ as_i\ as'_i$.)

Points (1)–(3) of Thm 100 model faithfully the indicated results from [Mazza 2012]. Together, they express that, for any position p, $[\![\_]\!]$ and $(\!|\_|\!)_p$ give mutually inverse bijections between terms

[8] Note that renaming equivalence is a partial equivalence on iterms, and an equivalence on uniform iterms.

and equivalence classes of uniform iterms w.r.t. renaming equivalence. And since Lemma 90 states that the actual renaming-equivalence representative produced by $(\!|\_|\!)_p$ is affine, the result can be read as establishing a *syntactic* isomorphism, up to renaming equivalence, between terms and uniform affine iterms.

Moreover, (4) and (5) of Thm 100 cover the *operational-semantics* component of the isomorphism, essentially stating that $[\![\_]\!]$ and $(\!|\_|\!)_p$ preserve (uniform) β-reduction. Recall from §E.3 that, in order to make Mazza's definition of uniform β-reduction rigorous, we had to define $\Rightarrow$ not on iterms like Mazza, but on streams on iterms. So while points (4) and (5) of our theorem correspond to Mazza's indicated results, they are not formulated exactly like those results—but they involve some lifting and mapping in order to work with streams of iterms. However, it is possible to recover Mazza's original formulations exactly. We do this by defining the inductive relation $\Rightarrow' : \mathit{ILTerm} \to \mathbb{N} \to \mathit{ILTerm} \to \mathit{Bool}$ as in Fig. 20. The definition of $\Rightarrow'$ matches Mazza's definition faithfully, in particular it does not commit to parallel reduction of streams of iterms until it becomes strictly necessary, namely for the right-application rule (iApR). As highlighted in the listing of (iApR), $\Rightarrow'$ makes use of our parallel relation $\Rightarrow$, i.e., from the moment one commits to parallel reduction one must stick to parallel reduction—which again, as far as we see, is the only way to make Mazza's definition rigorous.

To establish the formal connection between $\Rightarrow'$ (which is faithful to Mazza's definition) and $\Rightarrow$ (which is what allowed us to get the job done), the following result about $\Rightarrow$ is crucial. It states that, once $\Rightarrow_k$ has been established between two streams of terms ts and ts', it will be preserved no matter how we shuffle, duplicate or delete elements from ts and ts' in a synchronous manner, i.e., affecting the same positions in ts and ts':

Lemma 101. For all $ts, ts' \in \mathit{ILTerm}^\infty$, $k \in \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$, if $ts \Rightarrow_k ts'$ then $\mathit{map}\ (\lambda i.\ ts_{f\,i})\ (\mathit{natsFrom}\ 0) \Rightarrow_k \mathit{map}\ (\lambda i.\ ts'_{f\,i})\ (\mathit{natsFrom}\ 0)$.

(Note that the stream $\mathit{map}\ (\lambda i.\ ts_{f\,i})\ (\mathit{natsFrom}\ 0)$ consists of $ts_{f\,0}$, $ts_{f\,1}$, $ts_{f\,2}$ and so on.)

This result, which can be regarded as a form of equivariance, or more accurately parametricity of $\Rightarrow$ for stream indexes (w.r.t. arbitrary functions, not only small bijections), follows by standard rule induction. It has two important particular cases, where, for any item a, we write a^ω for the infinite stream that repeats a:

Lemma 102. The following hold:
(1) For all $ts, ts' \in \mathit{ILTerm}^\infty$ and $k \in \mathbb{N}$, if $ts \Rightarrow_k ts'$ then $\mathit{flat}\ ts^\omega \Rightarrow_k \mathit{flat}\ ts'^\omega$.
(2) For all $tss, tss' \in (\mathit{ILTerm}^\infty)^\infty$, $k \in \mathbb{N}$ and $i \in \mathbb{N}$, if $\mathit{flat}\ tss \Rightarrow_k \mathit{flat}\ tss'$ then $tss_i \Rightarrow_k tss'_i$.

These two particular cases allow us to connect $\Rightarrow'$ and $\Rightarrow$:

Lemma 103. The following hold:
(1) For all $t, t' \in \mathit{ILTerm}$ and $k \in \mathbb{N}$, if $t \Rightarrow'_k t'$ then $t^\omega \Rightarrow_k t'^\omega$.
(2) For all $ts, ts' \in \mathit{ILTerm}^\infty$, $k \in \mathbb{N}$ and $i \in \mathbb{N}$, if $ts \Rightarrow_k ts'$ then $ts_i \Rightarrow'_k ts'_i$.

Point (1) of Lemma 103 follows by rule induction using Lemma 102(1) in the (iApR) case; likewise, point (2) of Lemma 103 follows by rule induction using Lemma 102(2) in the (iApR) case.

Note that Lemma 103 implies an alternative definition of $\Rightarrow'$, namely $t \Rightarrow'_k t'$ iff $t^\omega \Rightarrow_k t'^\omega$, which further substantiates the intuition that in our development we worked with the parallelization of Mazza's relation. Using $\Rightarrow'$, we can now formulate Mazza-style the operational-semantics component of the isomorphism (i.e., reformulate points (4) and (5) of or Thm. 100):

Thm 104. The following hold:
(1) (Thm. 19(3) from [Mazza 2012]) For all $s, s' \in \mathit{LTerm}$, $k \in \mathbb{N}$ and $p \in \mathbb{N}^*$, if $s \to_k s'$ then there

$$\frac{t \Rightarrow_{\text{head}} t'}{t \Rightarrow'_0 t'} \text{(iBeta)} \qquad \frac{xs \in \mathit{Super} \qquad t \Rightarrow'_k t'}{\mathit{iLm}\ xs\ t \ \Rightarrow'_k \ \mathit{iLm}\ xs\ t'} \text{(iXi)}$$

$$\frac{\mathit{uniformS}\ ts \qquad t \Rightarrow'_k t'}{\mathit{iAp}\ t\ ts \ \Rightarrow'_{k+1} \ \mathit{iAp}\ t'\ tss} \text{(iApL)} \qquad \frac{\mathit{uniform}\ t \qquad ts \Rightarrow_k ts'}{\mathit{iAp}\ t\ ts \ \Rightarrow'_{k+1} \ \mathit{iAp}\ t\ ts'} \text{(iApR)}$$

Fig. 20. Uniform β-reduction for iterms

exists t' such that $[\![s]\!]_p \Rightarrow'_k t'$ and $t' \approx [\![s']\!]_p$ (in particular, *uniform* t').
(2) (Thm. 19(4) from [Mazza 2012]) For all $t, t' \in \mathit{ILTerm}$ and $k \in \mathbb{N}$, if $t \Rightarrow'_k t'$ then $(\!|t|\!) \to_k (\!|t'|\!)$.

The above follows immediately from Thm. 100(4,5) and Lemma 103. This concludes the statements of the isomorphism result from Mazza.

What we have not yet discussed is the proof of Thm. 100. We will do this next, highlighting as usual the places where strong rule induction was necessary. Point (1) of Thm. 100 follows by standard rule induction on $t \approx t'$, and point (2) by structural induction on the term s.

For point (3) of Thm. 100 , we use that "uniform implies good" and perform standard rule induction on *good* t; however, the uniformity assumption is also needed, i.e., the statement proved by rule induction is not

$$\mathit{good}\ t \text{ implies } \forall p.\ [\![(\!|t|\!)]\!]_p \approx t,$$

but

$$\mathit{good}\ t \text{ and } \mathit{uniform}\ t \text{ implies } \forall t.\ [\![(\!|t|\!)]\!]_p \approx t.$$

(Indeed, uniformity is needed in the (iApR) case.) Note also that using structural induction on t in conjunction with inversion rules for goodness or uniformity would not have been a valid alternative to rule induction, since in the abstraction case we would not have guaranteed that the binding stream of variables is a supervariable (as required for applying the corresponding inversion rule for uniformity, expressed by Lemma 81, or a similar inversion rule for goodness).

For proving points (4) and (5) of Thm. 100, it is clear that we need properties about the interaction between the translations and substitution. First, $[\![\_]\!]_\_$ commutes with substitution up to renaming equivalence, in the following way:

Lemma 105. (corresponds to Lemma 17 from [Mazza 2012]) For all $s, s' \in \mathit{LTerm}$, $p, q \in \mathbb{N}^*$, $qs \in (\mathbb{N}^*)^\infty$ and $x \in \mathit{Var}$, it holds that $[\![s[s'/x]]\!]_p \approx [\![s]\!]_q[(\mathit{map}\ [\![s']\!]_\_\ qs)\ /\ (\mathit{superOf}\ x)]$.

This lemma follows by strong structural induction on s—where the parameters' variables (to be avoided) are those of s' together with x.

Note that, thanks to the "positional" flexibility of $[\![\_]\!]_\_$ w.r.t. renaming equivalence (expressed by Lemma 87), in the above lemma we were able to allow on the right positions q, qs completely unrelated to the one on the left, p. (Mazza's Lemma 17 actually forces p and q to be equal, but this is unnecessary—which helps, because in the proof of Thm. 100(4) we need this stronger version. In addition, Mazza's Lemma 17 assumes that all positions in qs are mutually unrelated by the prefix order, which is also unnecessary.)

Now, point (4) of Thm. 100 follows by standard rule induction on $s \to_k s'$, using Lemma 105 in the (Beta) case (like Mazza anticipated).

As for $(\!|\_|\!)$, it also commutes with substitution, in the following sense:

Lemma 106. (Lemma 18 from [Mazza 2012]) For all $t \in \mathit{ILTerm}$, $ts \in \mathit{ILTerm}^\infty$ and $xs \in \mathit{Super}$ such that *uniform* t and *uniformS* ts, it holds that $(\!|t[ts/xs]|\!) = (\!|t|\!)[ts_0\ /\ (\mathit{superOf}^{-1}\ xs)]$.

To prove this, we again note (like when proving Thm. 100(3)) that uniformity implies goodness, which allows us to assume *good* t. And here again, structural induction on t would not do the job due to the impossibility of applying the inversion rule for uniformity or goodness. So the only option left is rule induction on *good* t; moreover, (unlike with Thm. 100(3)) this time we must cope with substitution, in particular avoid the variables in the abstraction case, so we need our strong rule induction on *good* t (Prop. 93)—which indeed gets the job done.

Finally, point (5) of Thm. 100 follows by standard rule induction on $ts \Rightarrow_k ts'$, using Lemma 106 in the (iBeta) case (again like Mazza anticipated).

E.9 Summary of the case study

Overall, our formal proofs were able to confirm Mazza's theorem that establishes an isomorphism between (finitary) λ-calculus under β-reduction and an infinitary uniform affine λ-calculus under a suitable notion of uniform β-reduction. Not only was Mazza right about the main theorem, but also his suggested sequence of lemmas leading to this main theorem were correct, and we ended up using them as Mazza envisioned.

Besides confirming the result, our formalization made a few contributions to rigor and clarity:

- We formally worked with terms and iterms modulo alpha-equivalence.
- We made the definition of uniform β-reduction rigorous, which required to shift from iterms to streams of iterms.
- We identified a few places in the lemmas where some of the assumptions made were unnecessary.
- As expected for a full detailed formal proof, there were quite a few gaps that needed to be filled.

Concerning the first point above, equating terms and iterms modulo α, this was explicitly Mazza's intention, as he writes in [Mazza 2012, § 2] (referring to the infinitary terms): "As usual, terms are always considered up to alpha-equivalence." Now, considering terms "up to" or "modulo" α usually means working with alpha-equivalence classes, which is what we did in our formalization.

Working with alpha-equivalence classes rather than "raw terms" has the huge benefit of substitution being well-behaved, which is essential in reasoning. But this also has a few drawbacks, the most important one being the higher difficulty of defining functions recursively. And indeed, for defining the translation operators we needed to deploy nominal recursors, which turned out to require a substantial formalization effort—especially since for translating uniform iterms to terms we ended up designing a custom recursor.

Rule inversion lemmas are another area where working with alpha-equivalence imposes constraints that may seem unintuitive at first. For example, one may hope to prove stronger versions of our inversion lemma for uniformity of abstractions, (Lemma 81), such as "if *uniform* (*iLm xs t*) then $xs \in$ *Super* and *uniform t*", or at least "if *uniform* (*iLm xs t*) then $xs \in$ *Super* or *uniform t*". But these do not hold on iterms as alpha-equivalence classes. This is because, since *iLm* is not injective, any iterm that has the form *iLm xs t* with *xs* supervariable and t uniform, also has the form *iLm ys s* where *ys* is not a supervariable and s is not uniform. This situation is imposing to formal developments like ours a certain discipline that is not visible from, and indeed is often bypassed by, informal developments—which, assuming alpha-equated terms for the sake of well-behaved substitution while also pretending that inversion rules à la free datatypes hold, want to have their cake and eat it too.

Another aspect where our formalization differs from Mazza's informal development is that, while he considers a countable set of variables for infinitary terms (iterms), we assume uncountably many. This is because, as noted by Blanchette et al. [Blanchette et al. 2019], as soon as we shift to infinitary terms, the usual assumptions that one usually takes for granted with finitary syntax no longer work here; and the countability of variables is one of these assumptions. Indeed, infinitary syntax makes it possible for a term to have an infinite number of free variables, running the danger of preventing the availability of (enough) fresh variables for it; in turn, this would negatively impact the definition of substitution, and even the very definition of alpha-equivalence that bootstraps the notion of terms as equivalence classes. Uncountably addresses this by making sure that we always have enough fresh variables. (We should note that other workarounds are sometimes possible. For example, Kurz et al. [Kurz et al. 2012, 2013] work with infinitary (coinductive) terms of finite support, which means allowing, for a term, infinitely many variables to participate in its bindings but only finitely many to appear free. This approach would not have worked here, since we need the iterms produced by the translation to have infinitely many distinct "copies" of the original free variables.)

Last but not least, we used this case study to validate this paper's main results, the strong rule induction principles. We have instantiated our general theorems to provide strong rule induction principles for the various notions of β-reduction, the *affine* predicate, renaming equivalence, and also the *good* predicate that emerged as an auxiliary to the infinitary-to-finitary translation. These principles have been used in key places in our proof development, mostly those involving the interaction between these different predicates and (finitary or infinitary) substitution.

F MORE DETAILS ON THE SYSTEM $F_{<:}$ SUBTYPING CASE STUDY

The POPLmark challenge [Aydemir et al. 2005] uses a presentation of subtyping that does not directly imply transitivity of subtyping (see figure 6). Proving this property is the goal of part one of the challenge. Similar to the case study in Appendix E, we have mechanized this theorem in Isabelle using our infrastructure.

As the syntax of System $F_{<:}$ is finite, the set of variables only need to be countable, i.e. of cardinality $\aleph_0$. Also, as seen in section 8.2, we first prove (by normal induction) that $\Gamma \vdash S <: T$ implies a well-formed context. This extra context allows us to derive the strong induction theorem for the subtyping predicate (Prop. 14) that will be used in the rest of this section.

Equipped with the strong induction theorem it is possible to directly follow the proof sketch outlined in the original POPLmark challenge. Instantiating the parameter structure of the strong induction theorem with the domain of the context ensures that the bound variable in the (All) case is not yet in the context. This directly allows to add the variable to the context while retaining well-formedness. Without this freshness we would need to manually rename the variables in all three arguments of the subtyping relation manually in ever proof.

The proof sketch starts out with reflexivity, context permutation and weakening for subtyping.

Lemma 107. If $wf\ \Gamma$ and $FV\ T \subseteq dom\,\Gamma$ then $\Gamma \vdash T <: T$

Lemma 108. (Permutation of the context) If $\Gamma \vdash S <: T$, $wf\ \Delta$ and $\Delta = \pi(\Gamma)$ then $\Delta \vdash S <: T$

Lemma 109. (Weakening of subtyping) If $\Gamma \vdash S <: T$ and $wf\ \Gamma, \Delta$ then $\Gamma, \Delta \vdash S <: T$

All these properties follow directly by strong induction, namely strong structural induction for Lemma 107 and strong rule induction on the definition of typing (Prop. 14) for Lemmas 108 and 109.

The most interesting case is the (All) case in the proof of Lemmas 109, where we use permutation to swap the new variable to the end of the context ($\Gamma, X <: T', \Delta$ to $\Gamma, \Delta, X <: T'$)

The proof of Lemma 107 follows by strong structural induction on the syntax of System $F_{<:}$, which is the following principle:

Prop 110. Let $(P, Psupp : P \to \mathcal{P}_{\mathsf{fin}}(Var))$ be a parameter structure. Let $\varphi : P \to Type \to Bool$ and assume that:

- (|TVr|): $\forall p, x.\ \varphi\ p\ (TVr\ X)$
- (|Top|): $\forall p.\ \varphi\ p\ Top$
- (|Arrow|): $\forall p, T_1, T_2.\ (\forall q.\ \varphi\ q\ T_1) \wedge (\forall q.\ \varphi\ q\ T_1) \longrightarrow \varphi\ p\ (T_1 \to T_2)$
- (|All|): $\forall \Gamma.\ (\forall \Gamma'.\ \varphi\ \Gamma'\ T_1) \wedge (\forall \Gamma'.\ \varphi\ \Gamma'\ T) \wedge X \notin FV\ \Gamma \longrightarrow \varphi\ \Gamma\ (\forall X <: T_1.\ T$

Then $\forall p, T.\ \varphi\ p\ T$.

Note that, like with all the (binding-aware) datatypes, the (strong) structural induction principle associated to a datatype (such as Prop. 110 for the datatype of System $F_{<:}$ syntax, Lemma 44 for the datatype of λ-calculus syntax, etc.) is a particular case of (strong) rule induction—namely, it coincides with the (strong) rule induction principle associated to an (alternative) inductive definition of equality on that datatype.

The proof of Lemma 107 shows the additional flexibility we get from the universal quantification over parameters. We use Prop. 110 where P is the set of contexts Γ and $Psupp\ \Gamma = FV\ \Gamma$. During the inductive proof, in the (|All|) case where the type has the form $\forall X <: T_1.\ T$, for a fixed Γ, we know that (1) $\Gamma' \vdash T_1 <: T_1$ for all Γ' and (2) $\Gamma' \vdash T <: T$ for all Γ', and must prove (3) $\Gamma \vdash (\forall X <: T_1.\ T) <: (\forall X <: T_1.\ T)$. And in order to prove the latter (using the rule (All) for typing from Fig. 6) we need to know that $\Gamma' \vdash T_1 <: T_1$ and $\Gamma, X <: T_1 \vdash T <: T$, so we must instantiate (1) with Γ and (2) with $(\Gamma, X <: T_1)$—so the universal quantification over the parameter was essential.

F.1 Transitivity and narrowing, version 1

With those basic lemmas it is now possible to prove transitivity of subtyping. However, as the proof sketch points out, transitivity requires narrowing which in turn requires transitivity (although only on smaller terms).

Thm 111. (Transitivity and Narrowing)
(1) $\Gamma \vdash S <: Q$ and $\Gamma \vdash Q <: T$ implies $\Gamma \vdash S <: T$
(2) $\Gamma, X <: Q, \Delta \vdash M <: N$ and $\Gamma \vdash R <: Q$ implies $\Gamma, X <: R, \Delta \vdash M <: N$

The proof uses simultaneous (strong) induction on Q followed by a (strong) rule induction on the resulting typing derivations. While the individual cases follow the exact steps that are described in the proof sketch, in the (All) case they require additional strong inversion rules, namely Lemma 112 and 113 below.

Lemma 112. (Strong rule inversion, first case)
If (1) $\Gamma \vdash (\forall X <: S_1.\ S_2) <: T$
(2) $X \notin dom\ \Gamma$
(3) for all Γ: $wf\ \Gamma$ and $FV\ (\forall X <: S_1.\ S_2) \subseteq dom\ \Gamma$ implies $P\ \Gamma\ (\forall X <: S_1.\ S_2)$ Top
(4) for all Γ, T_1 and T_2: $\Gamma \vdash T_1 <: S_1$ and $\Gamma, X <: T_1 \vdash S_2 <: T_2$ implies $P\ \Gamma\ (\forall X <: S_1.\ S_2)\ (\forall X <: T_1.\ T_2)$
then $P\ \Gamma\ (\forall X <: S_1.\ S_2); T$.

Lemma 113. (Strong rule inversion, second case)
If (1) $\Gamma \vdash S <: \forall X <: T_1.\ T_2$
(2) $X \notin dom\ \Gamma$
(3) for all Γ, Y, U: $Y <: U \in \Gamma$, $\Gamma \vdash U <: \forall X <: T_1.\ T_2$and $P\ \Gamma\ U\ (\forall X <: T_1.\ T_2)$ implies $P\ \Gamma\ (TVr\ Y)\ (\forall X <: T_1.\ T_2)$
(4) for all Γ, S_1, S_2: $\Gamma \vdash T_1 <: S_1$ and $\Gamma, X <: T_1 \vdash S_2 <: T_2$ implies $P\ \Gamma\ (\forall X <: S_1.\ S_2)\ (\forall X <: T_1.\ T_2)$
then $P\ \Gamma\ (\forall X <: S_1.\ S_2); T$.

These strong inversion rules allow to keep the exact same variable in the binder instead of obtaining a new one (as highlighted in their statement). To be able to derive these, the variable already needs to be fresh in the context. However, this is already the case thanks to the use of strong induction.

F.2 Transitivity and narrowing, version 2

While the individual cases of the nested, simultaneous induction in the previous section are straight forward, it requires a lot of very repetitive proof code. Given that narrowing only needs transitivity for strictly smaller terms, we can first prove narrowing by assuming transitivity:

Lemma 114. (Narrowing under assumed transitivity)
If (1) $\Gamma, X <: Q, \Delta \vdash M <: N$
(2) $\Gamma \vdash R <: Q$
(3) for all Γ,S, T: $\Gamma \vdash S <: Q$ and $\Gamma \vdash Q <: T$ implies $\Gamma \vdash S <: T$
then $\Gamma, X <: R, \Delta \vdash M <: N$

It is important that the type in the "middle" (highlighted above) is fixed, otherwise it would not be possible to use the induction hypothesis of the transitivity proof to fill in this assumption. Besides only proving a single theorem at a time, separating the proofs also allows to prove narrowing by induction on the typing derivation instead of on the type. This means that no rule inversions need to be done, further simplifying the proof. In fact, all but the (Trans-TV) case can be solved directly by the automation of Isabelle in our formalization.

With Lemma 114, it is possible to prove transitivity by induction on Q. For the (All) case the strong inversion rules are again useful to ensure that all binders use the same variable. As mentioned earlier, this case also uses the narrowing theorem by instantiating the extra assumption about transitivity with the induction hypothesis of the (All) case. After the proof is complete the full narrowing theorem can be obtain by plugging in the transitivity theorem.

G ISABELLE IMPLEMENTATION AND FORMALIZATION

G.1 Datatypes with bindings

We elaborate on our implementation (§10) of Blanchette et al.'s MRBNF-based foundational approach [Blanchette et al. 2019] to datatypes with bindings in Isabelle/HOL. It uses user-friendly custom syntax to define binder datatypes. For example, the type of λ-terms (App. D) can be introduced as follows in an Isabelle theory document:

```
binder_datatype 'var lterm =
  Var 'var
| App "'var lterm" "'var lterm"
| Lam x::'var t::"'var lterm" binds x in t
```

Internally this syntax creates the pre-datatype `lterm_pre`. This type distinguishes between bound and free positions and replaces recursive occurrences in the syntax with new type variables (distinguishing between recursive occurrences in which different variables are bound; here, nothing is bound in the arguments of `App`, whereas the first argument of `Lam` is bound in its second argument). The above declaration produces this pre-datatype:

```
('var, 'bvar, 'rec, 'brec) lterm_pre =
  'var
+ ('rec * 'rec)
+ ('bvar * 'brec)
```

The binding annotations written by the user are reflected in this type as the type variables representing recursive occurrences in the `App` and `Lam` cases are different. Furthermore the type has one free and one bound position (`'var` and `'bvar` respectively).

Afterwards the pre-datatype is proved to be an MRBNF by composition of known type constructors (in this case sum, product and identity). This composition also produces suitable `map` and free variable functions:

```
map_lterm_pre :: ('var => 'var) => ('bvar => 'bvar) => ('a => 'c) => ('b => 'd)
  => ('var, 'bvar, 'a, 'b) lterm_pre => ('var, 'bvar, 'c, 'd) lterm_pre
set1_lterm_pre :: ('var, 'bvar, 'rec, 'brec) lterm => 'var set
set2_lterm_pre :: ('var, 'bvar, 'rec, 'brec) lterm => 'bvar set
set3_lterm_pre :: ('var, 'bvar, 'rec, 'brec) lterm => 'rec set
set4_lterm_pre :: ('var, 'bvar, 'rec, 'brec) lterm => 'brec set
```

The fact that the pre-datatype is an MRBNF implies the existence of a (least) fixpoint. Its characteristic equation is:

```
'var lterm = ('var, 'var, 'var lterm, 'var lterm) lterm_pre
```

Solving the type fixpoint equation means here that the new type `'var LTerm` is defined as the quotient of the ordinary (non-binding) datatype

```
'var lterm_raw = 'var + 'var lterm_raw * 'var lterm_raw + 'var * 'var lterm_raw
```

by the alpha-equivalence relation induced by the binding relation. Moreover, the declaration defines constructors, substitution, and free variable functions, and proves their properties as described in App. D.1, e.g., distinctness, (quasi)-injectivity, and equivariance of the constructors, functorial properties of variable-for-variable substitution, and the strong structural induction principle. All these facts are proved automatically from first principles of Isabelle's higher-order logic.

Unlike in App. D.1, the introduced datatype is polymorphic in the variable type `'var`. This type variable is required to be large enough via a type class that the declaration introduces. In our example, `'var` is required to be infinite and can be thus instantiated with any infinite type; in our proofs we instantiate `'var` with (a type isomorphic to) `nat`.

The declaration can be easily adapted to yield λ-iterms (App. D.2). Note that the reference `xs` used to declare the binding relation for this complex binder appears nested in another type (`dstream`):

```
binder_datatype 'var iterm =
  Var 'var
| App "'var iterm" "'var iterm stream"
| Lam "(xs::'var) dstream" t::"'var iterm" binds xs in t
```

Here, `'a stream` is Isabelle's type of infinite sequences (defined as a codatatype in Isabelle's standard library) and `'a dstream` is a subtype of `'a stream` only containing sequences without repeating elements (which we have introduced specifically for this work). The new type's type variable `'var` is subject to a type class constraint that requires it to be uncountably infinite; in our proofs about λ-iterms we instantiate `'var` with an uncountable subtype of `nat set` (the type of sets of natural numbers).

Similar declarations yield types used in our other case studies: π-calculus (App. D.3) and System $F_{<:}$ (App. F). In general, our implementation supports multiple type variables and arbitrary constructor argument types. At the time of writing the user-friendly syntax does not directly support mutual recursion, however the underlying ML code covers this as well. We are in the process of providing support for binder codatatypes.

G.2 Rule induction

We formalize strong rule induction principles for inductive predicates using a hierarchy of locales [Ballarin 2014; Kammüller et al. 1999], Isabelle's module system. Specifically, locales constitute an extensible mechanism for managing local parameters and assumptions. For example, our first

```
binder_inductive step :: "lterm ⇒ lterm ⇒ bool" where
  Beta: "step (Ap (Lm x e1) e2) (tvsubst (Vr(x:=e2)) e1)"
| ApL: "step e1 e1' ⟹ step (Ap e1 e2) (Ap e1' e2)"
| ApR: "step e2 e2' ⟹ step (Ap e1 e2) (Ap e1 e2')"
| Xi: "step e e' ⟹ step (Lm x e) (Lm x e')" .

thm step.strong_induct step.equiv
```

☑ Proof state ☑ Auto update Update Search:

```
▪ step ?x1.0 ?x2.0 ⟹
  (⋀p. |?K p| <o |UNIV|) ⟹
  (⋀x e1 e2 p. x ∉ ?K p ⟹ ?P (Ap (Lm x e1) e2) (tvsubst (Vr(x := e2)) e1) p) ⟹
  (⋀e1 e1' e2 p. step e1 e1' ⟹ ∀p. ?P e1 e1' p ⟹ ?P (Ap e1 e2) (Ap e1' e2) p) ⟹
  (⋀e2 e2' e1 p. step e2 e2' ⟹ ∀p. ?P e2 e2' p ⟹ ?P (Ap e1 e2) (Ap e1 e2') p) ⟹
  (⋀e e' x p. x ∉ ?K p ⟹ step e e' ⟹ ∀p. ?P e e' p ⟹ ?P (Lm x e) (Lm x e') p) ⟹
  ∀p. ?P ?x1.0 ?x2.0 p
▪ bij ?σ ⟹
  |supp ?σ| <o |UNIV| ⟹ step ?x1.0 ?x2.0 ⟹ step (rrename ?σ ?x1.0) (rrename ?σ ?x2.0)
```

Fig. 21. Definition of β-reduction using binder_inductive.

variants of strong rule induction (Thms. 7 and 20) are proved abstractly in a locale called `Induct`, which (via distributed over several sublocales) fixes the structure of a κ-LS-nominal set $\mathcal{T}$ and the operator G and assumes the properties of being a κ-LS-nominal set, monotonicity, equivariance and $\mathcal{T}$-refreshability. The strong rule induction theorem is a statement about the inductive predicate I_G defined as the least fixpoint of G in the very same locale. A similar locale, for lack of a better name called `IInduct`, exists for our more general Thm. 23. The generality is captured by a sublocale relation showing that the parameters of `IInduct` can be instantiated using those of `Induct` (and suitable "passive" choices for the extra parameters of `IInduct`) and the assumptions of `IInduct` follow from the assumptions of `Induct` (given the above suitable choices). In principle, we could always work in the most general setting. However, the assumptions of `Induct` are easier to discharge for examples that do not need the full generality. In fact, we even defined and use even more restricted locale variants, called `Induct_simple` and `IInduct_simple`, which replace $\mathcal{T}$-refreshability with $\mathcal{T}$-freshness.

To obtain a concrete strong rule induction theorem for a conventional (non-binding-aware) inductive predicate I, a user can manually follow the following six steps. (1) Define the operator G, which underlies I's definition and abstracts over the bound variables, and (2) indicate the specific κ-LS-nominal set of interest. Then (3) instantiate (or interpret using Isabelle terminology) the `Induct` locale and, after (4) proving the locale's assumptions, obtain the principle about I_G. The syntactic mismatch between I and I_G is easily rectified by proving their equivalence: both are defined as least fixpoints of two operators that differ from each other only in that one abstracts over the bound variable positions and possibly some currying. (5) It is thus easy to prove $I = I_G$. Combining this fact with the strong rule induction about I_G, (6) another routine proof gives us the desired strong rule for I, where the inductive step is split into as many cases as there are introduction rules for I (whereas the rule for I_G only has one case formulated using G). The obtained rule is ready to be used with our binder_induction proof method.

Our binder_inductive and make_binder_inductive commands automate all these steps, while requiring the user to prove $\mathcal{T}$-refreshability in step (4). Moreover, the commands deviate from the above recipe in step (3): instead of instantiating the locale, the commands automate the proofs performed in the locale for the specific G. This is more convenient when dealing with currying: the locale's predicate I_G must always be uncurried and step (5) must rectify this mismatch in case the predicate I is curried, which is usually the case.

```
binder_inductive (no_auto_refresh) pstep :: "lterm ⇒ lterm ⇒ bool" where
  Refl: "pstep e e"
| Ap: "pstep e1 e1' ⟹ pstep e2 e2' ⟹ pstep (Ap e1 e2) (Ap e1' e2')"
| Xi: "pstep e e' ⟹ pstep (Lm x e) (Lm x e')"
| PBeta: "pstep e1 e1' ⟹ pstep e2 e2' ⟹ pstep (Ap (Lm x e1) e2) (tvsubst (Vr(x:=e2')) e1')"
  subgoal premises prems for R B t1 t2
    by (tactic ‹refreshability_tac false
      [@{term "FFVars :: lterm ⇒ var set"}, @{term "FFVars :: lterm ⇒ var set"}]
      [@{term "rrename :: (var ⇒ var) ⇒ lterm ⇒ lterm"},
       @{term "(λf x. f x) :: (var ⇒ var) ⇒ var ⇒ var"}]
      [NONE, NONE, SOME [SOME 0, SOME 0, SOME 1], SOME [SOME 0, SOME 0, NONE, NONE, SOME 1]]
      @{thm prems(3)} @{thm prems(2)} @{thms }
      @{thms emp_bound singl_bound ltermP.Un_bound ltermP.card_of_FFVars_bounds infinite}
      @{thms Lm_inject} @{thms Lm_eq_tvsubst ltermP.rrename_cong_ids[symmetric]}
      @{thms id_on_antimono} @{context}›)
  done

thm pstep.strong_induct pstep.equiv
```

Proof state Auto update Update Search: 100%

```
▪ pstep ?x1.0 ?x2.0 ⟹
  (⋀p. |?K p| <o |UNIV|) ⟹
  (⋀e p. ?P e e p) ⟹
  (⋀e1 e1' e2 e2' p.
      pstep e1 e1' ⟹
      ∀p. ?P e1 e1' p ⟹ pstep e2 e2' ⟹ ∀p. ?P e2 e2' p ⟹ ?P (Ap e1 e2) (Ap e1' e2') p) ⟹
  (⋀e e' x p. x ∉ ?K p ⟹ pstep e e' ⟹ ∀p. ?P e e' p ⟹ ?P (Lm x e) (Lm x e') p) ⟹
  (⋀e1 e1' e2 e2' x p.
      x ∉ ?K p ⟹
      pstep e1 e1' ⟹
      ∀p. ?P e1 e1' p ⟹
      pstep e2 e2' ⟹
      ∀p. ?P e2 e2' p ⟹ ?P (Ap (Lm x e1) e2) (tvsubst (Vr(x := e2')) e1') p) ⟹
  ∀p. ?P ?x1.0 ?x2.0 p
▪ bij ?σ ⟹
  |supp ?σ| <o |UNIV| ⟹ pstep ?x1.0 ?x2.0 ⟹ pstep (rrename ?σ ?x1.0) (rrename ?σ ?x2.0)
```

Fig. 22. Definition of parallel β-reduction using binder_inductive and using a semi-automated tactic for $\mathcal{T}$-refreshability.

As we discuss in §5 and §6, we have also taken steps to automate the required $\mathcal{T}$-refreshability proof. We have packaged the steps outlined in §6 as an ML tactic that orchestrates the retrieval of a fresh set of bound variables as well as the "as appropriate" instantiation of existential quantifiers and the following reasoning based on user input. Our ongoing work is synthesize this input automatically from the given introduction rules: at the moment of writing our automation succeeds in simple cases such as β-reduction, but still relies on user input for more complex examples. For example, Fig. 21 shows the full Isabelle script formalizing β-reduction and obtaining its equivariance and strong rule induction principle and Fig. 22 shows what needs to be done to obtain the same result for parallel β-reduction in which the $\mathcal{T}$-refreshability proof requires manual user input.

G.3 Statistics

Our implementation consists of 21 500 lines of Isabelle/ML, most of which are dedicated to the construction of MRBNF-based datatypes with bindings and recursive functions on such types. In addition, our formalization comprises of 16 000 lines of Isabelle definitions and proofs. Of those roughly 4 300 lines are dedicated to reusable infrastructure such as the formal prerequisites for the datatype construction, the locales for our enhanced rule induction principles, and generic theories for countable and uncountable variable types. The rest is distributed over our case studies: 1 200 lines for the formalization of System $F_{<:}$ and the proof of the POPLmark 1A challenge; 1 200 lines for the π-calculus formalization; 700 lines for the infinitary first-order logic; and 6 500 lines for the isomorphism between the λ-calculus and affine uniform infinitary λ-calculus. Our formalization contains 15 usages of strong rule induction principles and 17 usages of strong structural induction principles (always applied using the binder_induction proof method). Although the applications of strong induction principles are rare in absolute numbers, they were truly essential in our formalizations. For example, in the affine uniform infinitary λ-calculus case study we could follow Mazza's high-level proof sketches rather faithfully.