Journal of Automated Reasoning (2023) 67:23
https://doi.org/10.1007/s10817-023-09672-4

Rensets and Renaming-Based Recursion for Syntax with Bindings Extended Version

Andrei Popescu[1]

Received: 26 February 2023 / Accepted: 31 May 2023

Abstract
We introduce *renaming-enriched sets* (*rensets* for short), which are algebraic structures axiomatizing fundamental properties of renaming (also known as variable-for-variable substitution) on syntax with bindings. Rensets compare favorably in some respects with the well-known foundation based on nominal sets. In particular, renaming is a more fundamental operator than the nominal swapping operator and enjoys a simpler, equationally expressed relationship with the variable-freshness predicate. Together with some natural axioms matching properties of the syntactic constructors, rensets yield a truly minimalistic characterization of λ-calculus terms as an abstract datatype—one involving an infinite set of *unconditional equations*, referring only to the most fundamental term operators: the constructors and renaming. This characterization yields a recursion principle, which (similarly to the case of nominal sets) can be improved by incorporating Barendregt's variable convention. When interpreting syntax in semantic domains, our renaming-based recursor is easier to deploy than the nominal recursor. Our results have been validated with the proof assistant Isabelle/HOL.

Keywords Syntax with bindings · Renaming · Substitution · Recursion principle

1 Introduction

Formal reasoning about syntax with bindings is necessary for the meta-theory of logics, calculi and programming languages, and is notoriously error-prone. A great deal of research has been put into formal frameworks that make the specification of, and the reasoning about bindings more manageable.

Researchers wishing to formalize work involving syntax with bindings must choose a paradigm for representing and manipulating syntax—typically a variant of one of the "big three": nameful (sometimes called "nominal" reflecting its best known incarnation, nominal logic [27, 52]), nameless (De Bruijn) [4, 19, 62, 64] and higher-order abstract syntax (HOAS) [23, 24, 35, 45, 47]. Each paradigm has distinct advantages and drawbacks compared with

✉ Andrei Popescu
a.popescu@sheffield.ac.uk

[1] Department of Computer Science, University of Sheffield, Sheffield, UK

Published online: 05 July 2023

each of the others, some discussed at length, e.g., in [1, 13] and [31, §8.5]. And there are also hybrid approaches, which combine some of the advantages [17, 22, 55, 60].

A significant advantage of the nameful paradigm is that it stays close to the way one informally defines and manipulates syntax when describing systems in textbooks and research papers—where the binding variables are explicitly indicated. This can in principle ensure transparency of the formalization and allows the formalizer to focus on the high-level ideas. However, it only works if the technical challenge faced by the nameful paradigm is properly addressed: enabling the seamless definition and manipulation of concepts "up to alpha-equivalence", i.e., in such a way that the names of the bound variables are (present but nevertheless) inconsequential. This is particularly stringent in the case of recursion due to the binding constructors of terms not being free, hence not being a priori traversable recursively—in that simply writing some recursive clauses that traverse the constructors is not a priori guaranteed to produce a correct definition, but needs certain favorable conditions. The problem has been addressed by researchers in the form of tailored *nameful recursors* [27, 44, 52, 56, 68, 69], which are theorems that identify such favorable conditions and, based on them, guarantee the existence of functions that recurse over the non-free constructors.

In this paper, we make a contribution to the nameful paradigm in general, and to nameful recursion in particular. We introduce *rensets*, which are algebraic structures axiomatizing the properties of renaming, also known as variable-for-variable substitution, on terms with bindings (Sect. 3). Rensets differ from nominal sets (Sect. 2.2), which form the foundation of nominal logic, by their focus on (not necessarily injective) renaming rather than swapping (or permutation). Similarly to nominal sets, rensets are pervasive: Not only do variables and terms form rensets, but also any functor can lift the renset structure; thus, for example, the set of lists with elements from a renset, as well as the set of rose trees labelled with elements from a renset, are themselves rensets.

While lacking the pleasant symmetry of swapping, our axiomatization of renaming has its advantages. The first advantage is that renaming is more fundamental than swapping because, at an abstract axiomatic level, renaming can define swapping but not vice versa (Sect. 4). The second advantage is about the ability to define another central operator: the variable freshness predicate. While the definability of freshness from swapping is a signature trait of nominal logic, our renaming-based alternative fares even better: In rensets freshness has a simple, first-order definition (Sect. 3). This contrasts the nominal logic definition, which involves a second-order statement about (co)finiteness of a set of variables. The third advantage is largely a consequence of the second: Rensets enriched with constructor-like operators facilitate an equational characterization of terms with bindings (using an infinite set of unconditional equations), which does not seem possible for swapping (Sect. 5.1). This produces a recursion principle (Sects. 5.2 and 5.3) which, like the nominal recursor, caters for Barendregt's variable convention. We show that many functions on syntax can be defined using this recursion principle (Sect. 6). These include interpretations of syntax in semantics domains (Sect. 6.1), where this recursor is easier to apply than the nominal recursor. Our results have been mechanized in the Isabelle/HOL theorem prover [43] (Sect. 7), and will be integrated in an ongoing extension of Isabelle's (co)datatype package with binding-aware (co)datatypes [16].

In summary, we argue that our renaming-based axiomatization offers some benefits that strengthen the arsenal of the nameful paradigm: a simpler representation of freshness, a minimalistic equational characterization of terms, and a convenient recursion principle.

Here is the structure of the rest of this paper: Sect. 2 provides background on terms with bindings and on nominal logic. Section 3 introduces rensets and describes their basic properties. Section 4 establishes a formal connection to nominal sets. Section 5 discusses renaming-based recursion, and Sect. 6 shows example function definitions using renaming-

based recursion. Section 7 discusses our Isabelle formalization. Section 8 discusses related work.

For the novel results stated in this paper, we give pointers to the corresponding formalization in Isabelle—available from the Archive of Formal Proofs [58]. Each time, we indicate the name of the theory together with names of the relevant lemmas or theorems. More details on the mechanisms used in the formalization are given in Sect. 7.

This paper is an extension of our IJCAR 2022 conference paper [57].

2 Background

This section recalls the terms of λ-calculus and their basic operators (Sect. 2.1), and aspects of nominal logic including nominal sets and nominal recursion (Sect. 2.2).

2.1 Terms with Bindings

We work with the paradigmatic syntax of (untyped) λ-calculus. However, we believe our results can be generalized to syntaxes specified by arbitrary binding signatures such as the ones in [26, §2], [52, 71] or [16].

Let Var be a countably infinite set of variables, ranged over by x, y, z etc. The set Trm of *λ-terms* (or *terms* for short), ranged over by t, t_1, t_2 etc., is defined by the following grammar:

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

with the proviso that terms are equated (identified) modulo alpha-equivalence (also known as naming equivalence). Thus, e.g., if $x \neq z \neq y$ then $\mathsf{Lm}\ x\ (\mathsf{Ap}\ (\mathsf{Vr}\ x)\ (\mathsf{Vr}\ z))$ and $\mathsf{Lm}\ y\ (\mathsf{Ap}\ (\mathsf{Vr}\ y)\ (\mathsf{Vr}\ z))$ are considered to be the same term. We will often omit Vr when writing terms, as in, e.g., $\mathsf{Lm}\ x\ x$.

What the above specification means is (something equivalent to) the following: One first defines the set PTrm of *pre-terms* as freely generated by the grammar

$$p \ ::= \ \mathsf{PVr}\ x \ \mid \ \mathsf{PAp}\ p_1\ p_2 \ \mid \ \mathsf{PLm}\ x\ p$$

Then one defines the alpha-equivalence relation $\equiv : \mathsf{PTrm} \to \mathsf{PTrm} \to \mathsf{Bool}$ inductively, proves that it is an equivalence, and defines Trm by quotienting PTrm to alpha-equivalence, i.e., $\mathsf{Trm} = \mathsf{PTrm}/\equiv$. Finally, one proves that the pre-term constructors are compatible with $\equiv$, and defines the term counterpart of these constructors: $\mathsf{Vr} : \mathsf{Var} \to \mathsf{Trm}$, $\mathsf{Ap} : \mathsf{Trm} \to \mathsf{Trm} \to \mathsf{Trm}$ and $\mathsf{Lm} : \mathsf{Var} \to \mathsf{Trm} \to \mathsf{Trm}$.

The above constructions are technical, but well-understood, and can be fully automated for an arbitrary syntax with bindings (not just that of λ-calculus); and tools such as the Isabelle/Nominal package [71, 72] provide this automation, hiding pre-terms completely from the end user. In formal and informal presentations alike, one usually prefers to forget about pre-terms, and work with terms only. This has several advantages, including (1) being able to formalize concepts at the right abstraction level (since in most applications the naming of bound variables should be inconsequential) and (2) the renaming operator being well-behaved. However, there are some difficulties that need to be overcome when working with terms, and in this paper I focus on one of the major ones: providing recursion principles, i.e., mechanisms for defining functions by recursing over terms. This difficulty arises essentially because, unlike in the case of pre-term constructors, the binding constructor for terms is not free.

The main characters of our paper will be (generalizations of) some common operations and relations on Trm, namely:

- the constructors $\mathsf{Vr} : \mathsf{Var} \to \mathsf{Trm}$, $\mathsf{Ap} : \mathsf{Trm} \to \mathsf{Trm} \to \mathsf{Trm}$ and $\mathsf{Lm} : \mathsf{Var} \to \mathsf{Trm} \to \mathsf{Trm}$
- (capture-avoiding) renaming, also known as (capture-avoiding) substitution of variables for variables $\_[\_/\_] : \mathsf{Trm} \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Trm}$; for example, we have $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [x/y] = \mathsf{Lm}\ x'\ (\mathsf{Ap}\ x'\ x)$
- swapping $\_[\_\wedge\_] : \mathsf{Trm} \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Trm}$; for example, we have $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [x \wedge y] = \mathsf{Lm}\ y\ (\mathsf{Ap}\ y\ x)$
- the free-variable operator $\mathsf{FV} : \mathsf{Trm} \to \mathsf{Pow}(\mathsf{Var})$ (where $\mathsf{Pow}(\mathsf{Var})$ is the powerset of Var); for example, we have $\mathsf{FV}(\mathsf{Lm}\ x\ (\mathsf{Ap}\ y\ x)) = \{y\}$
- freshness $\_\#\_ : \mathsf{Var} \to \mathsf{Trm} \to \mathsf{Bool}$; for example, we have $x \,\#\, (\mathsf{Lm}\ x\ x)$; and assuming $x \neq y$, we have $\neg\ x \,\#\, (\mathsf{Lm}\ y\ x)$

The free-variable and freshness operators are of course related: A variable x is fresh for a term t (i.e., $x \,\#\, t$) if and only if it is not free in t (i.e., $x \notin \mathsf{FV}(t)$). The renaming operator $\_[\_/\_] : \mathsf{Trm} \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Trm}$ substitutes (in terms) *variables* for variables, not terms for variables.

Swapping of course makes sense not only in terms, but also in variables. Given variables x, y, z, $x[y \wedge z]$ is defined to be: z if $x = y$, y if $x = z$, and x otherwise.

2.2 Background on Nominal Logic

We will employ a formulation of nominal logic [51, 52, 69] that does not require any special logical foundation, e.g., axiomatic nominal set theory. For simplicity, we prefer the swapping-based formulation [51, 53] to the equivalent permutation-based formulation.

A *pre-nominal set* is a pair $\mathcal{A} = (A, \_[\_\wedge\_])$ where A is a set and $\_[\_\wedge\_] : A \to \mathsf{Perm} \to A$ is a function called *the swapping operator of* $\mathcal{A}$ satisfying the following properties for all $a \in A$ and $x, x_1, x_2, y_1, y_2 \in \mathsf{Var}$:

Identity:	$a[x \wedge x] = a$
Involution:	$a[x_1 \wedge x_2][x_1 \wedge x_2] = a$
Compositionality:	$a[x_1 \wedge x_2][y_1 \wedge y_2] = a[y_1 \wedge y_2][(x_1[y_1 \wedge y_2]) \wedge (x_2[y_1 \wedge y_2])]$

Given a pre-nominal set $\mathcal{A} = (A, \_[\_\wedge\_])$, an element $a \in A$ and a set $X \subseteq \mathsf{Var}$, one says that a *is supported by* X if $a[x \wedge y] = a$ holds for all $x, y \in \mathsf{Var}$ such that $x, y \notin X$. An element $a \in A$ is called *finitely supported* if there exists a finite set $X \subseteq A$ such that a is supported by X. A *nominal set* is a pre-nominal set $\mathcal{A} = (A, \_[\_\wedge\_])$ such that every element of a is finitely supported. If $\mathcal{A} = (A, \_[\_\wedge\_])$ is a nominal set and $a \in A$, then the smallest set $X \subseteq A$ such that a is supported by X exists, and is denoted by $\mathsf{supp}^{\mathcal{A}}\, a$ and called the *support of* a. One calls a variable x *fresh for* a, written $x \,\#\, a$, if $x \notin \mathsf{supp}^{\mathcal{A}}\, a$.

An alternative, more direct definition of freshness (which is preferred, e.g., by Isabelle/Nominal [71, 72]) is provided by the following proposition:

Prop 1 [72] For any nominal set $\mathcal{A} = (A, \_[\_\wedge\_])$ and any $x \in \mathsf{Var}$ and $a \in A$, it holds that $x \,\#\, a$ if and only if the set $\{y \mid a[y \wedge x] \neq a\}$ is finite.

Given two pre-nominal sets $\mathcal{A} = (A, \_[\_\wedge\_])$ and $\mathcal{B} = (B, \_[\_\wedge\_])$, the set $F = (A \to B)$ of functions from A to B becomes a pre-nominal set $\mathcal{F} = (F, \_[\_\wedge\_])$ by defining $f[x \wedge y]$ to send each $a \in A$ to $(f(a[x \wedge y]))[x \wedge y]$. $\mathcal{F}$ is not a nominal set because not all functions

are finitely supported (though of course one obtains a nominal set by restricting to finitely supported functions).

The set of terms together with their swapping operator, $(\mathsf{Trm}, \_[\_\wedge\_])$, forms a nominal set, where the support of a term is precisely its set of free variables. However, the power of nominal logic resides in the fact that not only the set of terms, but also many other sets can be organized as nominal sets—including the target domains of many functions one may wish to define on terms. This gives rise to a convenient mechanism for defining functions recursively on terms:

Theorem 2 [52] Let $\mathcal{A} = (A, \_[\_])$ be a nominal set and let $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$ and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ be some functions, all supported by a finite set X of variables and with $\mathsf{Lm}^{\mathcal{A}}$ satisfying the following freshness condition for binders (FCB): There exists $x \in \mathsf{Var}$ such that $x \notin X$ and $x \# \mathsf{Lm}^{\mathcal{A}}\ x\ a$ for all $a \in A$.

Then there exists a unique function $f : \mathsf{Trm} \to A$ that is supported by X and such that the following hold for all $x \in \mathsf{Var}$ and $t_1, t_2, t \in \mathsf{Trm}$:

(i) $f\ (\mathsf{Vr}\ x) = \mathsf{Vr}^{\mathcal{A}}\ x$
(ii) $f\ (\mathsf{Ap}\ t_1\ t_2) = \mathsf{Ap}^{\mathcal{A}}\ (f\ t_1)\ (f\ t_2)$
(iii) $f\ (\mathsf{Lm}\ x\ t) = \mathsf{Lm}^{\mathcal{A}}\ x\ (f\ t)$ if $x \notin X$

A useful feature of nominal recursion is the support for Barendregt's famous *variable convention* [11, p. 26]: "If [the terms] $t_1, \ldots, t_n$ 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." The above recursion principle adheres to this convention by fixing a finite set X of variables meant to be free in the definition context and guaranteeing that the bound variables in the definitional clauses are distinct from them. Formally, the target domain operators $\mathsf{Vr}^{\mathcal{A}}$, $\mathsf{Ap}^{\mathcal{A}}$ and $\mathsf{Lm}^{\mathcal{A}}$ are supported by X, and the clause for λ-abstraction is conditioned by the binding variable x being outside of X. (The Barendregt convention is also present in nominal logic via induction principles [52, 70–72].)

3 Rensets

This section introduces rensets, an alternative to nominal sets that axiomatize renaming rather than swapping or permutation.

A *renaming-enriched set* (*renset* for short) is a pair $\mathcal{A} = (A, \_[\_/\_])$ where A is a set and $\_[\_/\_] : A \to \mathsf{Var} \to \mathsf{Var} \to A$ is an operator such that the following hold for all $x, x_1, x_2, x_3, y, y_1, y_2 \in \mathsf{Var}$ and $a \in A$:

Identity:	$a[x/x] = a$
Idempotence:	If $x_1 \neq y$ then $a[x_1/y][x_2/y] = a[x_1/y]$
Chaining:	If $y \neq x_2$ then $a[y/x_2][x_2/x_1][x_3/x_2] = a[y/x_2][x_3/x_1]$
Commutativity:	If $x_2 \neq y_1 \neq x_1 \neq y_2$ then $a[x_2/x_1][y_2/y_1] = a[y_2/y_1][x_2/x_1]$

Let us call A the *carrier* of $\mathcal{A}$ and $\_[\_/\_]$ the *renaming operator* of $\mathcal{A}$. Similarly to the case of terms, we think of the elements $a \in A$ as some kind of variable-bearing entities and of $a[y/x]$ as the result of renaming x to y (i.e., substituting x with y) in a. With this intuition, the above properties are natural: Identity says that renaming a variable to itself has no effect. Idempotence acknowledges the fact that, after its renaming, a variable y is no longer there, so renaming it again has no effect. (Note that x_2 may be different from x_1 in the formulation of Idempotence, which means that this property is stronger than the operator $\_[x/y]$ being

idempotent for all variables x, y.) Chaining says that a chain of renamings $x_3/x_2/x_1$ has the same effect as the end-to-end renaming x_3/x_1 provided there is no interference from x_2, which is ensured by initially renaming x_2 to some other variable y. Finally, Commutativity allows the reordering of any two independent renamings.

Examples (see [58], theory Examples)
(Var, _[_/_]) and (Trm, _[_/_]), the sets of variables and terms with the standard renaming operator on them, form rensets. Moreover, given any functor F on the category of sets and a renset $\mathcal{A} = (A, \_[\_/\_])$, let us define the renset $F\,\mathcal{A} = (F\,A, \_[\_/\_])$ as follows: for any $k \in F\,A$ and $x, y \in \mathsf{Var}$, $k[x/y] = F\,(\_[x/y])\,k$, where the last occurrence of F refers to the action of the functor on morphisms.[1]

This means that one can freely build new rensets from existing ones using container types (which are particular kinds of functors)—e.g., lists, sets, trees etc.

In what follows, let us fix a renset $\mathcal{A} = (A, \_[\_/\_])$. One can define the notion of freshness of a variable for an element of a in the style of nominal logic. But the next proposition shows that simpler formulations are available.

Prop 3 (see [58], theory Rensets, lemmas freshA_iff_ex_vvsubstA_idle and freshA_iff_all_vvsubstA_idle) The following are equivalent:

(1) The set $\{y \in \mathsf{Var} \mid a[y/x] \neq a\}$ is finite.
(2) $a[y/x] = a$ for all $y \in \mathsf{Var}$.
(3) $a[y/x] = a$ for some $y \in \mathsf{Var} \setminus \{x\}$.

Let us define the predicate $\_\#\_ : \mathsf{Var} \to A \to \mathsf{Bool}$ as follows: $x \# a$, read *x is fresh for a*, if either of Prop. 3's equivalent properties holds.

Thus, points (1)–(3) above are three alternative formulations of $x \# a$, all referring to the lack of effect of renaming x to y, expressed as $a[y/x] = a$: namely that this phenomenon affects (1) all but a finite number of variables y, (2) all variables y, or (3) some variable $y \neq x$. The first formulation is the most complex of the three—it is the nominal definition, but using renaming instead of swapping. The other two formulations do not have counterparts in nominal logic, essentially because swapping is not as "efficient" as renaming at exposing freshness. In particular, (3) does not have a nominal counterpart because there is no single-swapping litmus test for freshness. The closest we can get to property (3) in a nominal set is the following: x is fresh for a if and only $a[y \wedge x] = a$ holds for some fresh y—but this needs freshness to explain freshness!

Examples (continued) For the rensets of variables and terms, freshness defined as above coincides with the expected operators: distinctness in the case of variables and standard freshness in the case of terms. And applying the definition of freshness to rensets obtained using finitary container types has similarly intuitive outcomes; for example, the freshness of a variable x for a list of items $[a_1, \ldots, a_n]$ means that x is fresh for each item a_i in the list.

Freshness satisfies some intuitive properties, which can be easily proved from its definition and the renset axioms. In particular, point (2) of the next proposition is the freshness-based version of the Chaining axiom.

[1] This is an instance of a more general phenomenon: For any equational theory $\mathcal{E}$ over a signature composed of unary operations only (like our renset operations $\_[x/y]$), and any functor F, we can lift any $\mathcal{E}$-algebra with carrier set A to an $\mathcal{E}$-algebra with carrier set $F\,A$. (And if the functor preserves products, we can do this for equational theories over any signatures, not restricted to unary operations only.)

Prop 4 (see [58], theory Rensets, lemmas freshA_vsubstA_idle, vsubstA_chain_freshA and freshA_vsubstA2)
The following hold:

(1) If $x \# a$ then $a[y/x] = a$
(2) $x_2 \# a$ then $a[x_2/x_1][x_3/x_2] = a[x_3/x_1]$
(3) If $z \# a$ or $z = x$, and $x \# a$ or $z \neq y$, then $z \# a[y/x]$

4 Connection to Nominal Sets

So far we focused on consequences of the purely equational theory of rensets, without making any assumption about cardinality. But after additionally postulating a nominal-style finite support property, one can show that rensets give rise to nominal sets—which is what we will do in this section.

Let us say that a renset $\mathcal{A} = (A, \_[\_/\_])$ has the *Finite Support* property if, for all $a \in A$, the set $\{x \in \mathsf{Var} \mid \neg\, x \# a\}$ is finite.

Let $\mathcal{A} = (A, \_[\_/\_])$ be a renset satisfying Finite Support. Let us define the swapping operator $\_[\_\wedge\_] : A \to \mathsf{Var} \to \mathsf{Var} \to A$ as $a[x_1 \wedge x_2] = a[y/x_1][x_1/x_2][x_2/y]$, where y is a variable that is fresh for all the involved items, namely $y \notin \{x_1, x_2\}$ and $y \# a$. Indeed, this is how one would define swapping from renaming on terms: using a fresh auxiliary variable y, and exploiting that such a fresh y exists and that its choice is immaterial for the end result. The next lemma shows that this style of definition also works abstractly, i.e., all it needs are the renset axioms plus Finite Support.

Lemma 5 (See [58], theory Rensets, lemma exists_freshA; and theory Rensets_to_Nominal_Sets, lemma vvsubstA_twoWays)
The following hold for all $x_1, x_2 \in \mathsf{Var}$ and $a \in A$:

(1) There exists $y \in \mathsf{Var}$ such that $y \notin \{x_1, x_2\}$ and $y \# a$.
(2) For all $y, y' \in \mathsf{Var}$ such that $y \notin \{x_1, x_2\}$, $y \# a$, $y' \notin \{x_1, x_2\}$ and $y' \# a$, $a[y/x_1][x_1/x_2][x_2/y] = a[y'/x_1][x_1/x_2][x_2/y']$.

And one indeed obtains an operator satisfying the nominal axioms:

Prop 6 (see [58], theory Rensets_to_Nominal_Sets, lemmas swapA_id, swapA_invol, swapA_cmp, freshA_swapA, and sublocale statement Renset_FinSupp < Nominal_Set)
If $(A, \_[\_/\_])$ is a renset satisfying Finite Support, then $(A, \_[\_\wedge\_])$ is a nominal set. Moreover, $(A, \_[\_/\_])$ and $(A, \_[\_\wedge\_])$ have the same notion of freshness, in that the freshness operator defined from renaming coincides with that defined from swapping.

The above construction is functorial, as we detail next. Given two nominal sets $\mathcal{A} = (A, \_[\_\wedge\_])$ and $\mathcal{B} = (B, \_[\_\wedge\_])$, a *nominal morphism* $f : \mathcal{A} \to \mathcal{B}$ is a function $f : A \to B$ with the property that it commutes with swapping, in that $(f\ a)[x \wedge y] = f(a[x \wedge y])$ for all $a \in A$ and $x, y \in \mathsf{Var}$. Nominal sets and nominal morphisms form a category that we will denote by $\underline{Nom}$. Similarly, let us define a morphism $f : \mathcal{A} \to \mathcal{B}$ between two rensets $\mathcal{A} = (A, \_[\_/\_])$ and $\mathcal{B} = (B, \_[\_])$ to be a function $f : A \to B$ that commutes with renaming, yielding the category $\underline{Ren}$ of rensets. Let us write $\underline{FRen}$ for the full subcategory of $\underline{Ren}$ given by rensets that satisfy Finite Support. Let us define $F : \underline{FRen} \to \underline{Nom}$ to be an operator on objects and morphisms that sends each finite-support renset to the above described nominal set constructed from it, and sends each renset morphism to itself. The reason why

the definition of F on morphisms is correct is the following: Let $\mathcal{A} = (A, \_[\_/\_])$ and $\mathcal{B} = (A, \_[\_/\_])$ be two finitely supported rensets, let $F\,\mathcal{A} = (A, \_[\_\wedge\_])$ and $F\,\mathcal{B} = (B, \_[\_\wedge\_])$, and assume $f : \mathcal{A} \to \mathcal{B}$ is a renset morphism. Then using the definition of freshness we obtain that, for all $x \in \mathsf{Var}$ and $a \in A$, $x \# a$ implies $x \# f(a)$. In turn, using the definition of $\_[\_ \wedge \_]$, this implies that f commutes with swapping (i.e., it is a nominal morphism). Indeed, for some y such that $y \notin \{x_1, x_2\}$ and $y \# a$, meaning that also $y \# f(a)$, we have that $f(a[x_1 \wedge x_2]) = f(a[y/x_1][x_1/x_2][x_2/y]) = f(a)[y/x_1][x_1/x_2][x_2/y] = f(a)[x_1 \wedge x_2]$.

One may ask whether it is also possible to make the trip back: from nominal sets to rensets. The answer is negative, at least if one wants to retain the same notion of freshness, i.e., have the freshness predicate defined in the nominal set be identical to the one defined in the resulting renset. This can be inferred from the fact that swapping preserves the cardinality of the support, whereas renaming must be allowed to change it since it might perform a non-injective renaming. The next example captures this idea:

Counterexample We will exhibit a nominal set $\mathcal{A} = (A, \_[\_\wedge\_])$, with freshness predicate $\$$ (defined from $\_[\_/\_]$ in nominal style as recalled in Section 2.2), such that there exists no operation $\_[\_/\_]$ on A satisfying the following two properties: (1) $(A, \_[\_/\_])$ is a renset with freshness predicate $\#$ (defined from $\_[\_/\_]$), and (2) $\$ = \#$. Namely, let $\mathcal{A} = (A, \_[\_\wedge\_])$ be a nominal set such that all elements of A have their support consisting of exactly two variables, x and y (with $x \neq y$). (For example, A can be the set of all terms with exactly these two free variables—this is indeed a nominal subset of the term nominal set because it is closed under swapping.) Assume for a contradiction that $\_[\_/\_]$ is an operation on A such that (1) and (2) hold. Then, by the definition of A, $a[y/x]$ needs to have exactly two variables z that are non-fresh variables (i.e., such that not $z \,\$\, a[y/x]$, or equivalently not $z \# a[y/x]$). But this is impossible, since by Prop. 4(3), all the variables different from y (including x) must be fresh for $a[y/x]$. In particular, $\mathcal{A}$ is not in the image of the functor $F : \underline{FRen} \to \underline{Nom}$, which is therefore not surjective on objects.

Thus, at an abstract algebraic level renaming can define swapping, but not the other way around. This is not too surprising, since swapping is fundamentally bijective whereas renaming is not; but it further validates our axioms for renaming, highlighting their ability to define a well-behaved swapping.

5 Recursion Based on Rensets

Prop. 3 shows that, in rensets, renaming can define freshness using only equality and universal or existential quantification over variables—without needing any cardinality condition like in the case of swapping. We show next that this forms the basis of a characterization of terms as the initial algebra of an equational theory (Sect. 5.1) and an expressive recursion principle—which we first describe in its simpler iterative form (Sect. 5.2), then as full-fledged primitive recursion (Sect. 5.3).

5.1 Equational Characterization of the Term Datatype

Rensets contain elements that are "term-like" in as much as there is a renaming operator on them satisfying familiar properties of renaming on terms. This similarity with terms can be strengthened by enriching rensets with operators having arities that match those of the term constructors.

A *constructor-enriched renset* (*CE renset* for short) is a tuple $\mathcal{A} = (A, \_[\_/\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ where:

- $(A, \_[\_/\_])$ is a renset
- $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$ and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ are functions

such that the following hold for all $a, a_1, a_2 \in A$ and $x, y, z \in \mathsf{Var}$:

(S1) $(\mathsf{Vr}^{\mathcal{A}}\ x)[y/z] = \mathsf{Vr}^{\mathcal{A}}(x[y/z])$
(S2) $(\mathsf{Ap}^{\mathcal{A}}\ a_1\ a_2)[y/z] = \mathsf{Ap}^{\mathcal{A}}(a_1[y/z])\ (a_2[y/z])$
(S3) if $x \notin \{y, z\}$ then $(\mathsf{Lm}^{\mathcal{A}}\ x\ a)[y/z] = \mathsf{Lm}^{\mathcal{A}}\ x\ (a[y/z])$
(S4) $(\mathsf{Lm}^{\mathcal{A}}\ x\ a)[y/x] = \mathsf{Lm}^{\mathcal{A}}\ x\ a$
(S5) if $z \neq y$ then $\mathsf{Lm}^{\mathcal{A}}\ x\ (a[z/y]) = \mathsf{Lm}^{\mathcal{A}}\ y\ (a[z/y][y/x])$

Let us call $\mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}}$ the *constructors* of $\mathcal{A}$. (S1)–(S3) express the constructors' commutation with renaming (with capture-avoidance provisions in the case of (S3)), (S4) the lack of effect of renaming a bound variable, and (S5) the possibility to rename a bound variable without changing the abstracted item (where the inner renaming of $z \neq y$ for y ensures the freshness of the "new name" y, hence its lack of interference with the other names in the "term-like" entity where the renaming takes place). All these are well-known to hold for terms:

Example (see [58], theory Examples)
Terms with renaming and the constructors, $(\mathsf{Trm}, \_[\_/\_], \mathsf{Vr}, \mathsf{Ap}, \mathsf{Lm})$, form a CE renset which will be denoted by $\mathcal{T}rm$.

As it turns out, the CE renset axioms capture exactly the term structure $\mathcal{T}rm$, via initiality. The notion of *CE renset morphism* $f : \mathcal{A} \to \mathcal{B}$ between two CE rensets $\mathcal{A} = (A, \_[\_/\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ and $\mathcal{B} = (B, \_[\_/\_], \mathsf{Vr}^{\mathcal{B}}, \mathsf{Ap}^{\mathcal{B}}, \mathsf{Lm}^{\mathcal{B}})$ is the expected one: a function $f : A \to B$ that is a renset morphism and also commutes with the constructors. Let us write $\underline{Ren}_{\mathsf{CE}}$ for the category of CE rensets and morphisms.

Theorem 7 (See [58], theory FRBCE_Rensets, theorems f_Vr, f_Ap, f_Lm, f_subst and f_unique; see also lemmas F_total and F_main in connection with the proof idea)
$\mathcal{T}rm$ is the initial CE renset, i.e., initial object in $\underline{Ren}_{\mathsf{CE}}$.

Proof idea. Let $\mathcal{A} = (A, \_[\_/\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ be a CE renset. Instead of directly going after a function $f : \mathsf{Trm} \to A$, one first inductively defines a relation $R : \mathsf{Trm} \to A \to \mathsf{Bool}$, with inductive clauses reflecting the desired properties concerning the commutation with the constructors, e.g., $\frac{R\ t\ a}{R\ (\mathsf{Lm}\ x\ t)\ (\mathsf{Lm}^{\mathcal{A}}\ x\ a)}$. It suffices to prove that R is total and functional and preserves renaming, since that allows one to define a constructor- and renaming-preserving function (a morphism) f by taking $f\ t$ to be the unique a with $R\ t\ a$.

Proving that R is total is easy by standard induction on terms. Proving the other two properties, namely functionality and preservation of renaming, is more elaborate and requires their simultaneous proof together with a third property: that R preserves freshness. The simultaneous three-property proof follows by a form of "renaming-based induction" on terms: Given a predicate $\varphi : \mathsf{Trm} \to \mathsf{Bool}$, to show $\forall t \in \mathsf{Trm}.\ \varphi\ t$ it suffices to show the following: (1) $\forall x \in \mathsf{Var}.\ \varphi\ (\mathsf{Vr}\ x)$, (2) $\forall t_1, t_2 \in \mathsf{Trm}.\ \varphi\ t_1\ \&\ \varphi\ t_2 \to \varphi\ (\mathsf{Ap}\ t_1\ t_2)$, and (3) $\forall x \in \mathsf{Var},\ t \in \mathsf{Trm}.\ (\forall s \in \mathsf{Trm}.\ \mathsf{Con}_{\_[\_/\_]}\ t\ s \to \varphi\ s) \to \varphi\ (\mathsf{Lm}\ x\ t)$, where $\mathsf{Con}_{\_[\_/\_]}\ t\ s$ means that t is connected to s by a chain of renamings.

Roughly speaking, R turns out to be functional because the λ-abstraction operator on the "term-like" inhabitants of A is, thanks to the axioms of CE renset, at least as non-injective as (i.e., identifies at least as many items as) the λ-abstraction operator on terms. □

Theorem 7 is the central result of this paper, from both practical and theoretical perspectives. Practically, it enables a useful form of recursion on terms (as we will discuss in the following sections). Theoretically, this is a characterization of terms as the initial algebra of an equational theory that only uses the most fundamental term operations, namely the constructors and renaming. The equational theory consists of the axioms of CE rensets [i.e., those of rensets plus (S1)–(S5)], which are an infinite set of unconditional equations—for example, axiom (S5) gives one equation for each pair of distinct variables y, z.

It is instructive to compare this characterization with the one offered by nominal logic, namely by Theorem 2. To do this, one first needs a lemma:

Lemma 8 [53] Let $f : A \to B$ be a function between two nominal sets $\mathcal{A} = (A, \_[\_\wedge\_])$ and $\mathcal{B} = (B, \_[\_\wedge\_])$ and X a set of variables. Then f is supported by X if and only if $f(a[x \wedge y]) = (f\, a)[x \wedge y]$ for all $x, y \in \mathsf{Var} \setminus X$.

Now Theorem 2 (with the variable avoidance set X taken to be $\emptyset$) can be rephrased as an initiality statement, as we describe below.

Let us define a *constructor-enriched nominal set* (*CE nominal set*) to be any tuple $\mathcal{A} = (A, \_[\_\wedge\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ where $(A, \_[\_\wedge\_])$ is a nominal set and $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$, $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ are operators on A such that the following properties hold for all $a, a_1, a_2 \in A$ and $x, y, z \in \mathsf{Var}$:

(N1) $(\mathsf{Vr}^{\mathcal{A}}\ x)[y \wedge z] = \mathsf{Vr}^{\mathcal{A}}(x[y \wedge z])$
(N2) $(\mathsf{Ap}^{\mathcal{A}}\ a_1\ a_2)[y \wedge z] = \mathsf{Ap}^{\mathcal{A}}(a_1[y \wedge z])\ (a_2[y \wedge z])$
(N3) $(\mathsf{Lm}^{\mathcal{A}}\ x\ a)[y \wedge z] = \mathsf{Lm}^{\mathcal{A}}\ (x[y \wedge z])\ (a[y \wedge z])$
(N4) $x \# \mathsf{Lm}\ x\ a$, i.e., $\{y \in \mathsf{Var} \mid (\mathsf{Lm}\ x\ a)[y \wedge x] \neq \mathsf{Lm}\ x\ a\}$ is finite.

The notion of *CE nominal morphism* is defined as the expected extension of that of nominal morphism: a function that commutes with swapping and the constructors. Let $\underline{Nom}_{\mathsf{CE}}$ be the category of CE nominal sets morphisms.

Theorem 9 ([52], rephrased) $(\mathsf{Trm}, \_[\_\ \wedge\ \_], \mathsf{Vr}, \mathsf{Ap}, \mathsf{Lm})$ is the initial CE nominal set, i.e., the initial object in $\underline{Nom}_{\mathsf{CE}}$.

The above theorem indeed corresponds exactly to Theorem 2 with $X = \emptyset$:

- the conditions (N1)–(N3) in the definition of CE nominal sets correspond (via Lemma 8) to the constructors being supported by $\emptyset$
- (N4) is the freshness condition for binders
- initiality, i.e., the existence of a unique morphism, is the same as the existence of the unique function $f : \mathsf{Trm} \to A$ stipulated in Theorem 2: commutation with the constructors is the Theorem 2 conditions (i)–(iii), and commutation with swapping means (via Lemma 8) f being supported by $\emptyset$.

Unlike the renaming-based characterization of terms (Theorem 7), the nominal logic characterization (Theorem 9) is not purely equational. This is due to a combination of two factors: (1) two of the axioms ((N4) and the Finite Support condition) referring to freshness and (2) the impossibility of expressing freshness equationally from swapping. We conjecture that the nominal characterization is not expressible purely equationally. By contrast, while the freshness idea is implicit in the CE renset axioms, the freshness predicate itself is absent from Theorem 7.

5.2 Barendregt-Enhanced Recursion Principle

While Theorem 7 already gives a recursion principle, it is possible to improve it by incorporating Barendregt's variable convention (in the style of Theorem 2):

Theorem 10 (See [58], theory FRBCE_Rensets, theorems f_Vr, f_Ap, f_Lm, f_subst and f_unique)
Let X be a finite set, $(A, \_[\_/\_])$ a renset and $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$ and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ some functions that satisfy the clauses (S1)–(S5) from the definition of CE renset, but only under the assumption that $x, y, z \notin X$.

Then there exists a unique function $f : \mathsf{Trm} \to A$ such that the following hold:

(i) $f\ (\mathsf{Vr}\ x) = \mathsf{Vr}^{\mathcal{A}}\ x$
(ii) $f\ (\mathsf{Ap}\ t_1\ t_2) = \mathsf{Ap}^{\mathcal{A}}\ (f\ t_1)\ (f\ t_2)$
(iii) $f\ (\mathsf{Lm}\ x\ t) = \mathsf{Lm}^{\mathcal{A}}\ x\ (f\ t)$ if $x \notin X$
(iv) $f\ (t[y/z]) = (f\ t)[y/z]$ if $y, z \notin X$

Proof idea. The constructions in the proof of Theorem 7 can be adapted to avoid clashing with the finite set of variables X. For example, the clause for λ-abstraction in the inductive definition of the relation R becomes $\frac{x \notin X \qquad R\ t\ a}{R\ (\mathsf{Lm}\ x\ t)\ (\mathsf{Lm}^{\mathcal{A}}\ x\ a)}$ and preservation of renaming and freshness are also formulated to avoid X. Totality is still ensured thanks to the possibility of renaming bound variables—in terms and inhabitants of A alike [via the modified axiom (S5)]. □

The above theorem says that if the structure $\mathcal{A}$ is assumed to be "almost" a CE renset, save for additional restrictions involving the avoidance of X, then there exists a unique "almost" CE renset morphism—satisfying the CE renset morphism conditions restricted so that the bound and renaming-participating variables avoid X. It is the renaming-based counterpart of the nominal Theorem 2.

In regards to the relative expressiveness of these two recursion principles (Theorems 10 and 2), it seems difficult to find an example that is definable by one but not by the other. In particular, our principle can seamlessly define standard nominal examples [52, 53] such as the length of a term, the counting of λ-abstractions or of the free-variables occurrences, and term-for-variable substitution—Sect. 6.3 gives details. However, as we will discuss in Sect. 6.1, we found an important class of examples where our renaming-based principle is significantly easier to deploy: that of interpreting syntax in semantic domains.

5.3 Full-Fledged Primitive Recursion

Theorem 10 restricts the recursive behavior to iteration, which allows the value of the defined function f on a term t to depend on the value of f on the components of t. For example, if t has the form $\mathsf{Ap}\ t_1\ t_2$, then $f\ t$ can depend on $f\ t_1$ and $f\ t_2$. This can routinely be extended to full primitive recursion, which additionally allows the dependence on the components themselves (not necessarily through f), e.g., on t_1 and t_2. Most recursors for syntax with bindings (including those we list later in Fig. 3) admit full primitive recursion enhancements. Our Theorem 10 is no exception—here is its enhancement, where we highlighted the additions:

A *full-recursion constructor-enriched renset* (*FRCE renset* for short) is a tuple $\mathcal{A} = (A, \_[\_/\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ where:

– $(A, \_[\_/\_])$ is a renset

– $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : \mathsf{Trm} \to A \to \mathsf{Trm} \to A \to A$, $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to \mathsf{Trm} \to A \to A$ are operators on A with arities matching those of the term constructors

such that the following properties hold for all $x, y, z \in \mathsf{Var} \setminus X$, $t, t_1, t_2 \in \mathsf{Trm}$ and $a, a_1, a_2 \in \mathsf{Trm}$:

(RS1) $(\mathsf{Vr}^{\mathcal{A}}\ x)[y/z] = \mathsf{Vr}^{\mathcal{A}}(x[y/z])$
(RS2) $(\mathsf{Ap}^{\mathcal{A}}\ t_1\ a_1\ t_1\ a_2)[y/z] = \mathsf{Ap}^{\mathcal{A}}\ (t_1[y/z])\ (a_1[y/z])\ (t_2[y/z])\ (a_2[y/z])$
(RS3) if $x \notin \{y, z\}$ then $(\mathsf{Lm}^{\mathcal{A}} x\ t\ a)[y/z] = \mathsf{Lm}^{\mathcal{A}} x\ (t[y/z])\ (a[y/z])$
(RS4) $(\mathsf{Lm}^{\mathcal{A}} x\ t\ a)[y/x] = \mathsf{Lm}^{\mathcal{A}}\ x\ t\ a$
(RS5) if $z \neq y$ then $\mathsf{Lm}^{\mathcal{A}} x\ (t[z/y])\ (a[z/y]) = \mathsf{Lm}^{\mathcal{A}} y\ (t[z/y][y/x])\ (a[z/y][y/x])$

Theorem 11 (See [58], theory FRBCE_Rensets, theorems f_Vr, f_Ap, f_Lm, f_subst and f_unique)
Let X be a finite set and $\mathcal{A} = (A, \_[\_/\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ be a FRCE renset. Then there exists a unique function $f : \mathsf{Trm} \to A$ such that the following hold:

(i) $f\ (\mathsf{Vr}\ x) = \mathsf{Vr}^{\mathcal{A}}\ x$
(ii) $f\ (\mathsf{Ap}\ t_1\ t_2) = \mathsf{Ap}^{\mathcal{A}}\ t_1\ (f\ t_1)\ t_2\ (f\ t_2)$
(iii) $f\ (\mathsf{Lm}\ x\ t) = \mathsf{Lm}^{\mathcal{A}}\ x\ t\ (f\ t)$ if $x \notin X$
(iv) $f\ (t[y/z]) = (f\ t)[y/z]$ if $y, z \notin X$

6 Example Functions Definable with the Renaming-Based Recursor

In this section, we show how to deploy our renaming-based recursor in function definitions. We start with a detailed discussion of semantic interpretation (Sect. 6.1), arguing that in this case renaming-based recursion fares better than the state-of-the-art nominal logic solution. Then we show other examples from the literature: some that fall under the semantic interpretation pattern (Sect. 6.2), and others that do not (Sect. 6.3). We believe that this varied list of examples evidences the wide versatility of renaming-based recursion.

6.1 Semantic Interpretation

Semantic interpretations, also known as denotations (or denotational semantics), are pervasive in the meta-theory of logics and λ-calculi, for example when interpreting first-order logic (FOL) formulas in FOL models, or untyped or simply-typed λ-calculus or higher-order logic terms in specific models (such as full-frame or Henkin models). In what follows, we will focus on λ-terms and Henkin models, but the ideas discussed apply broadly to any kind of statically scoped interpretation of terms or formulas involving binders.

Let D be a set and $\mathsf{ap} : D \to D \to D$ and $\mathsf{lm} : (D \to D) \to D$ be operators modeling semantic notions of application and abstraction. An environment will be a function $\xi : \mathsf{Var} \to D$. Given $x, y \in \mathsf{Var}$ and $d, e \in D$, let us write $\xi\langle x := d\rangle$ for ξ updated with value d for x (i.e., acting like ξ on all variables except for x where it returns d); and let us write $\xi\langle x := d, y := e\rangle$ instead of $\xi\langle x := d\rangle\langle y := e\rangle$.

Say one wants to interpret λ-terms in the semantic domain D in the context of environments, i.e., define the function $\mathsf{sem} : \mathsf{Trm} \to (\mathsf{Var} \to D) \to D$ that maps syntactic to semantic constructs; e.g., one would like to have:

– $\mathsf{sem}\ (\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ x))\ \xi = \mathsf{lm}(d \mapsto \mathsf{ap}\ d\ d)$ (regardless of ξ)

– $\mathsf{sem}\ (\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ \xi = \mathsf{lm}(d \mapsto \mathsf{ap}\ d\ (\xi\ y))$ (assuming $x \neq y$)

where we use $d \mapsto \ldots$ to describe functions in $D \to D$, e.g., $d \mapsto \mathsf{ap}\ d\ d$ is the function sending every $d \in D$ to $\mathsf{ap}\ d\ d$.

The definition should therefore naturally go recursively by the clauses:

(1) $\mathsf{sem}\ (\mathsf{Vr}\ x)\ \xi = \xi\ x$ (2) $\mathsf{sem}\ (\mathsf{Ap}\ t_1\ t_2)\ \xi = \mathsf{ap}\ (\mathsf{sem}\ t_1\ \xi)\ (\mathsf{sem}\ t_2\ \xi)$
(3) $\mathsf{sem}\ (\mathsf{Lm}\ x\ t)\ \xi = \mathsf{lm}\ (d \mapsto \mathsf{sem}\ t\ (\xi\langle x := d\rangle))$

Of course, since Trm is not a free datatype, these clauses do not work out of the box, i.e., do not form a definition (yet)—this is where binding-aware recursion principles such as Theorems 10 and 2 could step in. We will next try them both.

The three clauses above already determine constructor operations $\mathsf{Vr}^{\mathcal{I}}$, $\mathsf{Ap}^{\mathcal{I}}$ and $\mathsf{Lm}^{\mathcal{I}}$ on the set of interpretations, $I = (\mathsf{Var} \to D) \to D$, namely:

– $\mathsf{Vr}^{\mathcal{I}} : \mathsf{Var} \to I$ by $\mathsf{Vr}^{\mathcal{I}}\ x\ i\ \xi = \xi\ x$
– $\mathsf{Ap}^{\mathcal{I}} : I \to I \to I$ by $\mathsf{Ap}^{\mathcal{I}}\ i_1\ i_2\ \xi = \mathsf{ap}\ (i_1\ \xi)\ (i_2\ \xi)$
– $\mathsf{Lm}^{\mathcal{I}} : \mathsf{Var} \to I \to I$ by $\mathsf{Lm}^{\mathcal{I}}\ x\ i\ \xi = \mathsf{lm}\ (d \mapsto i\ (\xi\langle x := d\rangle))$

To apply the renaming-based recursion principle from Theorem 10, one must further define a renaming operator on I. Since the only chance to successfully apply this principle is if sem commutes with renaming, the definition should be inspired by the question: How can $\mathsf{sem}(t[y/x])$ be determined from $\mathsf{sem}\ t$, y and x? The answer is:

(4) $\mathsf{sem}\ (t[y/x])\ \xi = (\mathsf{sem}\ t)\ (\xi\langle x := \xi\ y\rangle)$,

yielding an operator $[\_/\_]^{\mathcal{I}} : I \to \mathsf{Var} \to \mathsf{Var} \to I$ defined by $i\ [y/x]^{\mathcal{I}}\ \xi = i\ (\xi\langle x := \xi\ y\rangle)$.

It is not difficult to verify that $\mathcal{I} = (I, [\_/\_]^{\mathcal{I}}, \mathsf{Vr}^{\mathcal{I}}, \mathsf{Ap}^{\mathcal{I}}, \mathsf{Lm}^{\mathcal{I}})$ is a CE renset—for example, Isabelle's automatic methods discharge all the goals. This means Theorem 10 (or, since here one doesn't need Barendregt's variable convention, already Theorem 7) is applicable, and gives us a unique function sem that commutes with the constructors, i.e., satisfies clauses (1)–(3) (which are instances of the clauses (i)–(iii) from Theorem 10), and additionally commutes with renaming, i.e., satisfies clause (4) (which is an instances of the clause (iv) from Theorem 10). (See [58], theory Examples.)

On the other hand, to apply nominal recursion for defining sem, one must identify a swapping operator on I. Similarly to the case of renaming, this identification process is guided by the goal of determining $\mathsf{sem}(t[x \wedge y])$ from $\mathsf{sem}\ t$, x and y, leading to (4′) $\mathsf{sem}\ (t[x \wedge y])\ \xi = \mathsf{sem}\ t\ (\xi\langle x := \xi\ y, y := \xi\ x\rangle)$, which yields the definition of $[\_ \wedge \_]^{\mathcal{I}}$ by $i\ [x \wedge y]^{\mathcal{I}}\ \xi = i\ (\xi\langle x := \xi\ y, y := \xi\ x\rangle)$. However, as pointed out by Pitts [52, §6.3] (in the slightly different context of interpreting simply-typed λ-calculus), the nominal recursor (Theorem 2) does *not* directly apply (hence neither does our reformulation based on CE nominal sets, Theorem 9). This is because, in our terminology, the structure $\mathcal{I} = (I, [\_ \wedge \_]^{\mathcal{I}}, \mathsf{Vr}^{\mathcal{I}}, \mathsf{Ap}^{\mathcal{I}}, \mathsf{Lm}^{\mathcal{I}})$ is not a CE nominal set. The problematic condition is FCB (the freshness condition for binders), requiring that $x\ \#^{\mathcal{I}}\ (\mathsf{Lm}^{\mathcal{I}}\ x\ i)$ holds for all $i \in I$. Expanding the definition of $\#^{\mathcal{I}}$ (the nominal definition of freshness from swapping, recalled in Sect. 2.2) and the definitions of $[\_\wedge\_]^{\mathcal{I}}$ and $\mathsf{Lm}^{\mathcal{I}}$, one can see that $x\ \#^{\mathcal{I}}\ (\mathsf{Lm}^{\mathcal{I}}\ x\ i)$ means the following:

$\mathsf{lm}\ (d \mapsto i\ (\xi\langle x := \xi\ y, y := \xi\ x\rangle\langle x := d\rangle)) = \mathsf{lm}\ (d \mapsto i\ (\xi\langle x := d\rangle))$, i.e., $\mathsf{lm}\ (d \mapsto i\ (\xi\langle x := d, y := \xi\ x\rangle) = \mathsf{lm}\ (d \mapsto i\ (\xi\langle x := d\rangle))$, holds for all but a finite number of variables y.

The only chance for the above to be true is if i, when applied to an environment, ignores the value of y in that environment for all but a finite number of variables y; in other words, i only analyzes the value of a finite number of variables in that environment—but this is not

guaranteed to hold for arbitrary elements $i \in I$. To repair this, Pitts engages in a form of induction–recursion [21], carving out from I a smaller domain that is still large enough to interpret all terms, then proving that both FCB and the other axioms hold for this restricted domain. It all works out in the end, but the technicalities are quite involved.

Although FCB is not required by the renaming-based principle, note incidentally that this condition would actually be true (and immediate to check) if working with freshness defined not from swapping but from renaming. Indeed, the renaming-based version of $x \#^{\mathcal{I}} (\mathsf{Lm}^{\mathcal{I}}\ x\ i)$ says that $\mathsf{lm}\ (d \mapsto i\ (\xi\langle x := \xi\ y\rangle\langle x := d\rangle)) = \mathsf{lm}\ (d \mapsto i\ (\xi\langle x := d\rangle))$ holds for all y (or at least for some $y \neq x$)—which is immediate since $\xi\langle x := \xi\ y\rangle\langle x := d\rangle = \xi\langle x := d\rangle$. This further illustrates the idea that semantic domains 'favor' renaming over swapping.

In conclusion, for interpreting syntax in semantic domains, our renaming-based recursor is trivial to apply, whereas the nominal recursor requires some fairly involved additional definitions and proofs.

6.2 Two Instances of the Semantic Interpretation Pattern

For the next two examples, taken from the literature on HOAS and nominal logic, we deploy the semantic interpretation solution discussed in Sect. 6.1. Indeed, the notion of semantic domain can be chosen flexibly, to also cover certain purely syntactic operators as well. (For the formalization of these two examples, see [58], theory Examples.)

6.2.1 Number of Bound Variables

Consider the task of defining an operator $\mathsf{cbv} : \mathsf{Trm} \to \mathbb{N}$ that counts the number of bound variables occurring in a term. In his book [53], Pitts defines it following the approach of Schürmann et al. [63] as $\mathsf{cbv}\ t = \mathsf{cbvs}\ t\ (x \mapsto 0)$, where the auxiliary function $\mathsf{cbvs} : \mathsf{Trm} \to (\mathsf{Var} \to \mathbb{N}) \to \mathbb{N}$ operates according the following recursive clauses:

(1) $\mathsf{cbvs}\ (\mathsf{Vr}\ x)\ \xi = \xi\ x$
(2) $\mathsf{cbvs}\ (\mathsf{Ap}\ t_1\ t_2)\ \xi = (\mathsf{cbvs}\ t_1\ \xi) + (\mathsf{cbvs}\ t_2\ \xi)$
(3) $\mathsf{cbvs}\ (\mathsf{Lm}\ x\ t)\ \xi = \mathsf{cbvs}\ t\ (\xi\langle x := 1\rangle)$

Thus, this situation is seen as a particular case of semantic interpretation. (The same complications as with semantic interpretation arise with deploying the nominal recursor, and Pitts deploys a similar workaround.)

6.2.2 Eta-Reducibility Checking

When illustrating his parametric HOAS approach, Chlipala defines a function $\mathsf{canEta} : \mathsf{Term} \to \mathsf{Bool}$ that checks whether a term can eta expand (i.e., is an eta-redex) [18, Fig. 3]. canEta is defined as follows:

$$\mathsf{canEta}\ t = \begin{cases} \mathsf{true}\ , \text{ if } t \text{ has the form } \mathsf{Lm}\ x\ (\mathsf{Ap}\ s\ x) \\ \qquad\qquad \text{and } \mathsf{canEta}'\ s\ (\top\langle x := \mathsf{false}\rangle) = \mathsf{true} \\ \mathsf{false}\ , \text{ otherwise} \end{cases}$$

where $\top$ is the environment sending all variables to true and $\mathsf{canEta}' : \mathsf{Term} \to (\mathsf{Var} \to \mathsf{Bool}) \to \mathsf{Bool}$ is such that $\mathsf{canEta}'\ t\ \xi$ checks whether the variables that are assigned false in the environments are fresh for the term t. Chlipala defines 'canEta' using parametric HOAS,

which seems to essentially equivalent to the semantic domain interpretation pattern. In this case, the definitional clauses for canEta′, translated into our formalism, are the following:

(1) $\mathsf{canEta}'\ (\mathsf{Vr}\ x)\ \xi = \xi\ x$
(2) $\mathsf{canEta}'\ (\mathsf{Ap}\ t_1\ t_2)\ \xi = (\mathsf{canEta}'\ t_1\ \xi)\ \&\ (\mathsf{canEta}'\ t_2\ \xi)$
(3) $\mathsf{canEta}'\ (\mathsf{Lm}\ x\ t)\ \xi = \mathsf{canEta}'\ t\ (\xi\langle x := \mathsf{true}\rangle)$

canEta can of course be alternatively defined by other means, e.g., using the free-variable operator.

6.3 Other Examples

Next we show a few other examples, purely syntactic in nature. For each of them, we will not indicate the required CE renset or BCE renset. Rather, we show the clauses describing the behavior of the defined function with respect to the constructors and renaming—from these, the corresponding structure on the target domain can be easily inferred, like we did in Sect. 6.1. In each case, the verification of the necessary properties to deploy our Theorem 10 is trivial. (See [58], theory Examples.)

The length of a term [52, Example 4.2], $\mathsf{length} : \mathsf{Term} \to \mathbb{N}$.

- $\mathsf{length}\ (\mathsf{Vr}\ x) = 1$
- $\mathsf{length}\ (\mathsf{Ap}\ t_1\ t_2) = \mathsf{max}\ (\mathsf{length}\ t_1, \mathsf{length}\ t_2) + 1$
- $\mathsf{length}\ (\mathsf{Lm}\ x\ t) = \mathsf{length}\ t + 1$
- $\mathsf{length}\ (t\ [x/y]) = \mathsf{length}\ t$

Counting λ-abstractions [53, Example 8.18], $\mathsf{clam} : \mathsf{Term} \to \mathbb{N}$.

- $\mathsf{clam}\ (\mathsf{Vr}\ x) = 0$
- $\mathsf{clam}\ (\mathsf{Ap}\ t_1\ t_2) = \mathsf{clam}\ t_1 + \mathsf{clam}\ t_2$
- $\mathsf{clam}\ (\mathsf{Lm}\ x\ t) = \mathsf{clam}\ t + 1$
- $\mathsf{clam}\ (t\ [x/y]) = \mathsf{clam}\ t$

Counting the number of free occurrences of a variable, $\mathsf{cfv} : \mathsf{Term} \to \mathsf{Var} \to \mathbb{N}$.

- $\mathsf{cfv}\ (\mathsf{Vr}\ y)\ x = (\text{if } x = y \text{ then } 1 \text{ else } 0)$
- $\mathsf{cfv}\ (\mathsf{Ap}\ t_1\ t_2)\ x = \mathsf{cfv}\ t_1\ x + \mathsf{cfv}\ t_2\ x$
- $\mathsf{cfv}\ (\mathsf{Lm}\ y\ t)\ x = (\text{if } x = y \text{ then } 0 \text{ else } \mathsf{cfv}\ t\ x)$
- $\mathsf{cfv}\ (t[z/y])\ x = \begin{cases} \mathsf{cfv}\ t\ x\ , \text{if } x \notin \{y, z\} \\ \mathsf{cfv}\ t\ x + \mathsf{cfv}\ t\ y\ , \text{if } x = z \neq y \\ 0\ , \text{if } x = y \neq z \\ \mathsf{cfv}\ t\ y\ , \text{if } x = y = z \end{cases}$

Term-for-variable substitution [53, Example 8.18], $\_[\![\_/\_]\!] : \mathsf{Trm} \to \mathsf{Trm} \to \mathsf{Var} \to \mathsf{Trm}$.

- $(\mathsf{Vr}\ y)\ [\![s/x]\!] = \begin{cases} s\ , \text{if } x = y \\ \mathsf{Vr}\ y\ , \text{otherwise} \end{cases}$
- $(\mathsf{Ap}\ t_1\ t_2)\ [\![s/x]\!] = \mathsf{Ap}\ (t_1\ [\![s/x]\!])\ (t_2\ [\![s/x]\!])$
- $(\mathsf{Lm}\ y\ t)\ [\![s/x]\!] = \mathsf{Lm}\ y\ (t\ [\![s/x]\!])$ if $y \notin \{x\} \cup \mathsf{FV}\ s$
- $t\ [y/z]\ [\![s/x]\!] = t\ [\![s/x]\!]\ [y/z]$ if $y, z \notin \{x\} \cup \mathsf{FV}\ s$

(So here one applies the recursion principle with the Barendregt parameter X taken to be $\{x\} \cup \mathsf{FV}\ s$.)

Defining (capture-avoiding) parallel substitution, $\_[\![\_]\!] : \mathsf{Trm} \to (\mathsf{Var} \to_{\mathsf{fin}} \mathsf{Trm}) \to \mathsf{Var} \to \mathsf{Trm}$, where $\mathsf{Var} \to_{\mathsf{fin}} \mathsf{Trm}$ is the set of functions $\rho : \mathsf{Var} \to \mathsf{Trm}$ having $\mathsf{supp}\ \rho$ finite (where $\mathsf{supp}\ \rho$ consists of all variables x such that $\rho\ x \neq \mathsf{Vr}\ s$), is similar:

- $(\mathsf{Vr}\ y)\ [\![\rho]\!] = \rho\ y$
- $(\mathsf{Ap}\ t_1\ t_2)\ [\![\rho]\!] = \mathsf{Ap}\ (t_1\ [\![\rho]\!])\ (t_2\ [\![\rho]\!])$
- $(\mathsf{Lm}\ y\ t)\ [\![\rho]\!] = \mathsf{Lm}\ y\ (t\ [\![\rho]\!])$ if $y \notin \mathsf{supp}\ \rho$
- $t\ [y/z]\ [\![\rho]\!] = t\ [\![\rho]\!]\ [y/z]$ if $y, z \notin \mathsf{supp}\ \rho$ and $y, z \notin \mathsf{FV}\ (\rho\ x)$ for all $x \in \mathsf{Var}$.

(So here one applies the recursion principle with the Barendregt parameter X taken to be $\mathsf{supp}\ \rho \cup \{\mathsf{FV}\ (\rho\ x) \mid x \in \mathsf{supp}\ \rho\}$.)

7 Isabelle Formalization

Our Isabelle formalization of this paper's results is available from the Archive of Formal Proofs [58]. Next we will describe the overall structure of this formalization, after briefly recalling the locale modularization mechanism—which has been instrumental in the formalization.

7.1 Primer on Isabelle's Locales

The formalization makes heavy use of locales [9, 40], which are an elegant mechanism for managing structures and assumptions in Isabelle. A locale fixes some types, constants and assumptions. One can perform definitions and prove theorems inside a locale, and everything happens relative to the entities fixed in that locale. Viewed from outside the locale, all these definitions and theorems are (1) polymorphic in that locale's fixed types, (2) universally quantified over that locale's constants, and (3) conditioned by that locale's assumptions. A locale can extend other locales (thus inheriting all their types, constants and assumptions).

A locale can be *interpreted* at the top level (of an Isabelle theory) by providing concrete types and constants for that locale's fixed types and constants, and verifying the locale's assumptions; after a successful interpretation, all the definitions performed and theorems proved in a locale are automatically instantiated for these concrete types and constants. A locale L_1 can also be interpreted relative to another locale L_2 by establishing a *sublocale* relationship $L_1 < L_2$. This amounts to showing that the entities of L_1 can provide an interpretation for those of L_2; i.e., assuming the fixed types, constants and assumptions of L_1, one identifies some types and constants that instantiate those of L_2, and verifies the assumptions of L_2.

As can be seen from the above description, locales are useful for developing the mathematics of algebraic structures, such as groups, rings, fields etc. Then (top-level) interpretations provide particular examples of such structures, e.g., interpreting the ring locale into the particular ring of integers. Moreover, sublocale relationships are useful for showing the inclusion between two types of structures, e.g., fields are particular kinds of rings, or more generally to show that one type of structure induces another type of structure.

7.2 Overview of the Formalization

The Isabelle/HOL formalization of this paper's results and examples has the theory structure shown in Fig. 1, where the theories have self-explanatory names. It consists of 2000

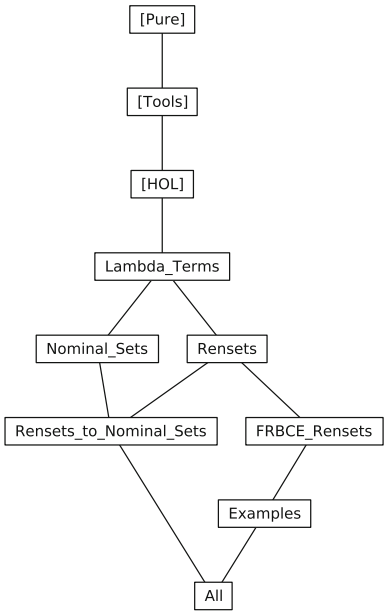

Fig. 1 The Isabelle theories

LOC in total, of which 900 LOC are dedicated to preliminary results on syntax (theory Lambda_Terms), 900 to the results leading to the theorems on comparison with nominal sets and recursion principles, and 200 LOC on examples (theory Examples).

Locales have been instrumental in keeping our development structured and concise. We have introduced the following locales:

- Renset, which fixes the type A and the renaming operator _[_/_] and assumes the renset axioms.
- Renset_FinSupp, which includes Renset and adds the finite support assumption.
- Renset_Morphism, which includes two copies of Renset_FinSupp, say with fixed types A and B, and fixes a function $f : A \to B$ assumed to be a renset morphism.
- Pre_Nominal_Set, which fixes the type A and the swapping operator _[_∧_] and assumes the pre-nominal set axioms.
- Nominal_Set extends Pre_Nominal_Set with the Finite Support axiom, obtaining the nominal set axioms.
- Nominal_Morphism, which is similar to Renset_Morphism, but for expressing nominal set morphisms on top of two copies of Nominal_Set.
- CE_Renset, which includes Renset and adds the additional constructor operators and assumptions of CE rensets.
- BCE_Renset (read "Barendregt CE renset"), which includes Renset and adds a set of variables X and the additional constructor operators and assumptions of "CE rensets up to the avoidance of X", i.e., the assumptions of Theorem 10.
- FRBCE_Renset (read "full-recursion Barendregt CE renset"), which includes Renset and adds the constants and assumptions from Theorem 11 (the extension of Theorem 10 to full-fledged recursion described in Sect. 5.3).

```
locale Renset =
fixes vsubstA :: "'A ⇒ var ⇒ var ⇒ 'A"
assumes
vsubstA_id[simp]: "⋀x a. vsubstA a x x = a"
and
vsubstA_idem[simp]: "⋀x y1 y2 a. y1 ≠ x ⟹ vsubstA (vsubstA a y1 x) y2 x = vsubstA a y1 x"
and
vsubstA_chain: "⋀u x1 x2 x3 a.
  u ≠ x2 ⟹
  vsubstA (vsubstA (vsubstA a u x2) x2 x1) x3 x2 =
  vsubstA (vsubstA a u x2) x3 x1"
and
vsubstA_commute_diff:
"⋀ x y u a v. x ≠ v ⟹ y ≠ u ⟹ x ≠ y ⟹
vsubstA (vsubstA a u x) v y = vsubstA (vsubstA a v y) u x"
```

Fig. 2 The Isabelle locale Renset

– Sem_Int and Local_Functor, which are locales used for two of our examples (discussed below).

For example, Fig. 2 show the Isabelle locale Renset which axiomatizes rensets. It fixes a type 'A (modelled as a type variable) and a renaming (variable-for-variable substitution) operator on it called vsubstA (which in this paper is denoted by _[_/_]), and assumes the renset properties (two of which are also declared as simplification rules for reasoning, via the "simp" attribute). All the facts described in Sect. 3 are proved in the Renset locale.

The connection between rensets satisfying Finite Support and nominal sets described in Sect. 4 is worked out inside the locales Renset and Renset_Morphism (namely, the swapping operator is defined, its properties are inferred from the renset axioms, etc.) and finalized in the form of the sublocale relationships Renset $<$ Nominal_Set and Renset_Morphism $<$ Nominal_Morphism.

The initiality/recursion principle for rensets from Sect. 5 is proved in the most general locale, FRBCE_Renset, which features both the Barendregt and the full-fledged recursion enhancement. In other words, we prove the most general theorem Theorem 11. Then Theorems 10 and 7, which are particular cases, are obtained along the sublocale relationships:

$$\mathsf{CE\_Renset} < \mathsf{BCE\_Renset} < \mathsf{FRBCE\_Renset}$$

(Strictly speaking, we don't need the less general locales CE_Renset and BCE_Renset, but we kept them in this archive for better documenting the results reported in the main paper.)

The examples of rensets from Sect. 3 and of recursive definitions from Sect. 6 are mostly formalized as interpretations of the relevant locales, for example, Renset and CE_Renset are instantiated to the type of terms and its standard operators. There are two exceptions to this rule. The first is the example of constructing a renset from a functor and another renset, which is handled via a sublocale relationship: Local_Functor + Renset $<$ Renset, where the lefthand side is a locale that puts together the assumptions of an operator F acting as a functor on functions $A \to A$ and of A with a renaming operator being a renset.[2] The other is the semantic interpretation example discussed in Sect. 6.1, which is itself abstract (in that it works with unspecified operators ap and lm) and therefore is formalized via a locale Sem_Int which fixes these operators and a sublocale relationship Sem_Int $<$ CE_Renset. For all the examples defined with our recursor, the recursion theorem's assumptions are discharged

[2] We call F a "local functor" because functoriality is only assumed with respect to functions on A; taking this weaker assumption allows me to model the example using a locale where we fix the type $F\ A$ along with A; thus bypassing the problem of locales not allowing to be parameterized by type constructors.

automatically by Isabelle with the help of the internal automation (auto, and friends) or the external automation (Sledgehammer [46]). This brings some experimental evidence for the ease of deploying the recursor.

8 Conclusion and Related Work

This paper introduced and studied rensets, contributing (1) theoretically, minimalistic equational characterizations of the datatype of terms with bindings and (2) practically, an addition to the formal arsenal for manipulating syntax with bindings. It is part of a longstanding line of work by ourselves and collaborators on exploring convenient definition and reasoning principles for bindings [31, 34, 56, 59, 60], and will be incorporated into the ongoing implementation of a new Isabelle definitional package for binding-aware datatypes [16].

8.1 Initial Model Characterizations of the Terms Datatype

Our results provide a truly elementary characterization of terms with bindings, as an "ordinary" datatype specified by the fundamental operations only (the constructors plus renaming) and some equations (those defining CE rensets). As far as specification simplicity goes, this is "the next best thing" after a completely free datatype such as those of natural numbers or lists.

Figure 3 shows previous characterizations from the literature, in which terms with bindings are identified as an initial model (or algebra) of some kind. For each of these, we indicate (1) the employed reasoning paradigm, (2) whether the initiality/recursion theorem features an extension with Barendregt's variable convention, (3) the underlying category (from where the carriers of the models are taken), (4) the operations and relations on terms to which the models must provide counterparts and (5) the properties required on the models.

While some of these results enjoy elegant mathematical properties of intrinsic value, our main interest is in the recursors they enable, specifically in the ease of deploying these recursors. That is, we are interested in how easy it is in principle to organize the target domain as a model of the requested type, hence obtain the desired morphism, i.e., get the recursive definition done. By this measure, elementary approaches relying on standard FOL-like models whose carriers are sets rather than pre-sheaves have an advantage. Also, it seems intuitive that a recursor is easier to apply if there are fewer operators, and fewer and structurally simpler properties required on its models—although empirical evidence of successfully deploying the recursor in practice should complement the simplicity assessment, to ensure that simplicity is not sponsored by lack of expressiveness.

The first column in the upper half of Fig. 3's table contains an influential representative of the nameless paradigm: the result obtained independently by Fiore et al. [26] and Hofmann [36] characterizing terms as initial in the category of algebras over the pre-sheaf topos $Set^{\mathbb{F}}$, where $\mathbb{F}$ is the category of finite ordinals and functions between them. The operators required by algebras are the constructors, as well as the free-variable operator (implicitly as part of the separation on levels) and the injective renamings (as part of the functorial structure). The algebra's carrier is required to be a functor and the constructors to be natural transformations. There are several variations of this approach, e.g., [5, 15, 36], some implemented in proof assistants, e.g., [3, 4, 39].

The other columns (in both the upper half and lower half of the figure) refer to initiality results that are more closely related to ours. They take place within the nameful paradigm,

	Fiore et al. [26] Hofmann [36]	Pitts [51]	Urban et al. [69,68]	Norrish [43]
Paradigm	nameless	nameful	nameful	nameful
Barendregt?	n/a	yes	yes	yes
Underlying category	$Set^{\mathbb{F}}$	Set	Set	Set
Required operations/ relations	ctors, rename, free-vars	ctors, perm	ctors, perm	ctors, swap, free-vars
Required properties	functori-ality, naturality	Horn clauses, fresh-def, fin-supp	Horn clauses, fresh-def	Horn clauses

	Popescu & Gunter [58]	Gheri & Popescu [31]	This paper
Paradigm	nameful	nameful	nameful
Barendregt?	no	no	yes
Underlying category	Set	Set	Set
	ctors, term/var subst, fresh	ctors, swap, fresh	ctors, rename
Required properties	Horn clauses	Horn clauses	equations

Fig. 3 Initial model characterizations of the datatype of terms with bindings "ctors" = "constructors", "perm" = "permutation", "fresh" = "the freshness predicate", "fresh-def" = "clause for defining the freshness predicate", "fin-supp" = "Finite Support"

and they all rely on elementary models (with set carriers). Pitts's already discussed nominal recursor [52] (based on previous work by Gabbay and Pitts [27]) employs the constructors and permutation (or swapping), and requires that its models satisfy some Horn clauses for constructors, permutation and freshness, together with the second-order properties that (1) define freshness from swapping and (2) express Finite Support. Urban et al.'s version [68, 69] implemented in Isabelle/Nominal is an improvement of Pitts's in that it removes the Finite Support requirement from the models—which is practically significant because it enables non-finitely supported target domains for recursion. Norrish's result [44] is explicitly inspired by nominal logic, but renounces the definability of the free-variable operator from swapping—with the price of taking both swapping and free-variables as primitives. Our previous work with Gunter and Gheri takes as primitives either term-for-variable substitution and freshness [59] or swapping and freshness [31], and requires properties expressed by different Horn clauses (and does not explore a Barendregt dimension, like Pitts, Urban et al. and Norrish do). Our previous focus on term-for-variable substitution [59] (as opposed to renaming, i.e., variable-for-variable substitution) impairs expressiveness—for example, the depth of a term is not definable using a recursor based on term-for-variable substitution because we cannot say how term-for-variable substitution affects the depth of a term based on its depth and that of the substitutee alone. Our results based on rensets keep freshness out of the primitive operators base (like nominal logic does), and provide unconditionally equational characterizations using

only constructors and renaming.[3] The key to achieving this minimality is the simple expression of freshness from renaming in our axiomatization of rensets. In future work, we plan a systematic formal comparison of the relative expressiveness of all these nameful recursors.

8.2 Recursors in Other Paradigms

Figure 3 focuses on nameful recursors, while considering only the Fiore et al./Hofmann recursor for the sake of a rough comparison with the nameless approach. We should stress that such a comparison is necessarily rough, since the nameless recursors do not give the same "payload" as the nameful ones. This is because of the handling of bound variables. In the nameless paradigm, the λ-constructor does not explicitly take a variable as an input, as in $\mathsf{Lm}\ x\ t$, i.e., does not have type $\mathsf{Var} \to \mathsf{Trm} \to \mathsf{Trm}$. Instead, the bindings are indicated through nameless pointers to positions in a term. So the nameless λ-constructor, let's call it NLm, takes only a term, as in $\mathsf{NLm}\ t$, i.e., has type $\mathsf{Trm} \to \mathsf{Trm}$ or a scope-safe (polymorphic or dependently-typed) variation of this, e.g., $\prod_{n\in\mathbb{F}} \mathsf{Trm}_n \to \mathsf{Trm}_{n+1}$ [26, 36] or $\prod_{\alpha\in\mathsf{Type}} \mathsf{Trm}_\alpha \to \mathsf{Trm}_{\alpha+\mathsf{unit}}$ [5, 15]. The λ-constructor is of course matched by operators in the considered models, which appears in the clauses of the functions f defined recursively on terms: Instead of a clause of the form $f\ (\mathsf{Lm}\ x\ t) = \langle$expression depending on x and $f\ t\rangle$ from the nameful paradigm, in the nameless paradigm one gets a clause of the form $f\ (\mathsf{NLm}\ t) = \langle$ expression depending on $f\ t\rangle$. A nameless recursor is usually easier to prove correct and easier to apply because the nameless constructor NLm is free—whereas a nameful recursor must wrestle with the non-freeness of Lm, handled by verifying certain properties of the target models. However, once the definition is done, having nameful clauses pays off by allowing "textbook-style" proofs that stay close to the informal presentation of a calculus or logic, whereas with the nameless definition some additional index shifting bureaucracy is necessary. (See [13] for a detailed discussion.) Hybrid nameless/nominal solutions have also been proposed, notably the locally named [42, 55] and locally nameless [7, 17] representations.

A comparison of nameful recursion with HOAS recursion is also generally difficult, since major HOAS frameworks such as Abella [8], Beluga [49] or Twelf [48] are developed within non-standard logical foundations, allowing a λ-constructor of type $(\mathsf{Trm} \to \mathsf{Trm}) \to \mathsf{Trm}$, which is not amenable to typical well-foundedness based recursion but requires some custom solutions (e.g., [25, 63]). However, the *weak HOAS* variant [20, 34] employs a constructor of the form $\mathsf{WHLm} : (\mathsf{Var} \to \mathsf{Trm}) \to \mathsf{Trm}$ which *is* recursable, and in fact yields a free datatype, let us call it WHTrm—one generated by $\mathsf{WHVr} : \mathsf{Var} \to \mathsf{WHTrm}$, $\mathsf{WHAp} : \mathsf{WHTrm} \to \mathsf{WHTrm} \to \mathsf{WHTrm}$ and WHLm. WHTrm contains (natural encodings of) all terms but also additional entities referred to as "exotic terms". Partly because of the exotic terms, this free datatype by itself is not very helpful for recursively defining useful functions on terms. But the situation is dramatically improved if one employs a variant of weak HOAS called *parametric HOAS (PHOAS)* [18], i.e., takes Var not as a fixed type but as a type parameter (type variable) and works with $\prod_{\mathsf{Var}\in\mathsf{Type}} \mathsf{Trm}_{\mathsf{Var}}$; this enables many useful definitions by choosing a suitable type Var (usually large enough to make the necessary distinctions) and then performing standard recursion. The functions definable in the style of PHOAS seem to be exactly those definable via the semantic domain interpretation pattern (Sect. 6.1): Choosing

[3] One could say that De Bruijn representations achieve an even higher level of "purity", namely absolute freeness as opposed to freeness modulo an equational theory—but for a different binding constructor, and with the price of either dealing with dangling references or moving away from the category of sets; see also Sect. 8.2.

the instantiation of Var to a type T corresponds to employing environments in $\mathsf{Var} \to T$. (Our Sect. 6.3 illustrates this by showing the semantic-domain version of a PHOAS example.)

As a hybrid nameful/HOAS approach we can count Gordon and Melham's characterization of the datatype of terms [33], which employs the nameful constructors but formulates recursion treating Lm as if recursing in the weak-HOAS datatype WHTrm. Norrish's recursor [44] (a participant in Fig. 3) has been inferred from Gordon and Melham's one. Weak-HOAS recursion also has interesting connections with nameless recursion: In presheaf toposes such as those employed by Fiore et al. [26], Hofmann [36] and Ambler et al. [6], for any object T the function space $\mathsf{Var} \Rightarrow T$ is isomorphic to the De Bruijn level shifting transformation applied to T; this effectively equates the weak-HOAS and nameless recursors. A final cross-paradigm note: In themselves, nominal sets are not confined to the nameful paradigm; their category is equivalent [27] to the Schanuel topos [38], which is attractive for pursuing the nameless approach.

8.3 Axiomatizations of Renaming

In his study of name-passing process calculi, Staton [65] considers an enrichment of nominal sets with renaming (in addition to swapping) and axiomatizes renaming with the help of the nominal (swapping-defined) freshness predicate. He shows that the resulted category is equivalent to the non-injective renaming counterpart of the Schanuel topos (i.e., the subcategory of $\mathit{Set}^{\mathbb{F}}$ consisting of functors that preserve pullbacks of monos). Gabbay and Hofmann [28] provide an elementary characterization of the above category, in terms of *nominal renaming sets*, which are sets equipped with a multiple-variable-renaming action satisfying identity and composition laws, and a form of Finite Support (FS). Nominal renaming sets are very related to rensets satisfying FS. Indeed, any nominal renaming set forms a FS-satisfying renset when restricted to single-variable renaming; and conversely, any FS-satisfying renset gives rise to a nominal renaming set. (This relationship, which we conjectured in the conference version of this paper [57], was proved recently by Pitts [50], see below.) This correspondence seems similar to the one between the permutation-based and swapping-based alternative axiomatizations of nominal sets—in that the two express the same concept up to an isomorphism of categories. In their paper, Gabbay and Hofmann do not study renaming-based recursion, beyond noting the availability of a recursor stemming from the functor-category view (which, as we discussed above, enables nameless recursion with a weak-HOAS flavor). Pitts [54] introduces *nominal sets with* 01*-substitution structure*, which axiomatize substitution of one of two possible constants for variables on top of the nominal axiomatization, and proves that they form a category that is equivalent with that of cubical sets [14], hence relevant for the univalent foundations [37].

In recent work [50], Pitts introduces *locally nameless sets*, an algebraic axiomatization of syntax under the locally nameless representation, and characterizes the locally nameless recursor [17] using initiality in a functor category (similarly to recursors in the nameless setting [26, 36]). He also proves that the category of locally nameless sets is equivalent to that of finitely supported rensets and to the aforementioned categories considered by Staton [65] and Gabbay and Hofmann [28]. On the way to his results, Pitts gives an alternative axiomatization of finitely supported rensets, using instead of the Chaining axiom a simpler (unconditional) axiom: $t[x_2/x_1][x_3/x_2] = t[x_3/x_2][x_3/x_1]$.

8.4 Early Insight into Renaming and Substitution

In the context of formalizing pure type systems [10, 12], McKinna and Pollack [41, 42] emphasize closure under renamings as instrumental in reasoning modulo alpha. In follow-up work, Goguen and McKinna [32] also reduce closure under substitution to closure under renamings. Stoughton [66] proposes defining parallel substitution on pre-terms by primitive recursion; in this approach, bound-variable captures are avoided by performing renamings not on the pre-term but on the substitution function (similarly to how semantic interpretation proceeds using environments).

8.5 Other Work

Sun [67] develops universal algebra for first-order languages with bindings (generalizing work by Aczel [2]) and proves a completeness theorem. Gabbay and Mathijssen [29] axiomatize term-for-variable substitution in nominal logic and use it to develop an extension of first-order logic that internalizes some meta-level concepts [30]. In joint work with Popescu and Roşu [61], we develop first-order logic and prove completeness on top of a generic syntax with axiomatized free-variables and substitution.

8.6 Renaming Versus Swapping and Nominal Logic, Final Round

We believe that our work complements rather than competes with nominal logic. Our results do not challenge the swapping-based approach to defining syntax (defining the alpha-equivalence on pre-terms and quotienting to obtain terms) recommended by nominal logic, which is more elegant than a renaming-based alternative; but our easier-to-apply recursor can be a useful addition even on top of the nominal substratum. Moreover, some of our constructions are explicitly inspired by the nominal ones. For example, We started by adapting the nominal idea of defining freshness from swapping before noticing that renaming enables a simpler formulation. Our formal treatment of Barendregt's variable convention also originates from nominal logic—as it turns out, this idea works equally well in our setting. In fact, We came to believe that the possibility of a Barendregt enhancement is largely orthogonal to the particularities of a binding-aware recursor. In future work, we plan to investigate this, i.e., seek general conditions under which an initiality principle (such as Theorems 9 and 7) is amenable to a Barendregt enhancement (such as Theorems 2 and 10, respectively).

Acknowledgements I am grateful to the anonymous IJCAR and JAR reviewers, and to Andy Pitts and James McKinna, for their insightful comments and suggestions, and for pointing out relevant results and related work, and to Dmitriy Traytel for suggesting the short name "renset".

References

1. Abel, A., Allais, G., Hameer, A., Pientka, B., Momigliano, A., Schäfer, S., Stark, K.: POPLMark reloaded: mechanizing proofs by logical relations. J. Funct. Program. **29**, e19 (2019). https://doi.org/10.1017/S0956796819000170
2. Aczel, P.: Frege structures and notations in propositions, truth and set. In: The Kleene Symposium, pp. 31–59. North Holland (1980)
3. Allais, G., Atkey, R., Chapman, J., McBride, C., McKinna, J.: A type and scope safe universe of syntaxes with binding: their semantics and proofs. In: Proceedings of ACM Programming Languages 2(International Conference on Functional Programming (ICFP)), 2018, pp. 90:1–90:30 (2018). https://doi.org/10.1145/3236785
4. Allais, G., Chapman, J., McBride, C., McKinna, J.: Type-and-scope safe programs and their proofs. In: Bertot, Y., Vafeiadis, V. (eds.) Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, 16–17 January 2017. pp. 195–207. ACM (2017). https://doi.org/10.1145/3018610.3018613
5. Altenkirch, T., Reus, B.: Monadic presentations of lambda terms using generalized inductive types. In: Flum, J., Rodríguez-Artalejo, M. (eds.) Computer Science Logic (CSL) 1999. LNCS, vol. 1683, pp. 453–468 (1999). https://doi.org/10.1007/3-540-48168-0_32
6. Ambler, S.J., Crole, R.L., Momigliano, A.: A definitional approach to primitivexs recursion over higher order abstract syntax. In: Eighth ACM SIGPLAN International Conference on Functional Programming, Workshop on Mechanized Reasoning About Languages with Variable Binding, MERLIN 2003, Uppsala, Sweden, August 2003. ACM (2003). https://doi.org/10.1145/976571.976572
7. Aydemir, B.E., Charguéraud, A., Pierce, B.C., Pollack, R., Weirich, S.: Engineering formal metatheory. In: Necula, G.C., Wadler, P. (eds.) Principles of Programming Languages (POPL) 2008, pp. 3–15. ACM (2008). https://doi.org/10.1145/1328438.1328443
8. Baelde, D., Chaudhuri, K., Gacek, A., Miller, D., Nadathur, G., Tiu, A., Wang, Y.: Abella: a system for reasoning about relational specifications. J. Formaliz. Reason. **7**(2), 1–89 (2014). https://doi.org/10.6092/issn.1972-5787/4650
9. Ballarin, C.: Locales: a module system for mathematical theories. J. Autom. Reason. **52**(2), 123–153 (2014). https://doi.org/10.1007/s10817-013-9284-7
10. Barendregt, H.P., Dekkers, W., Statman, R.: Lambda Calculus with Types. Perspectives in Logic. Cambridge University Press (2013). http://www.cambridge.org/de/academic/subjects/mathematics/logic-categories-and-sets/lambda-calculus-types
11. Barendregt, H.P.: The Lambda Calculus: Its Syntax and Semantics, Studies in Logic, vol. 40. Elsevier, Amsterdam (1984)
12. Berardi, S.: Towards a Mathematical Analysis of the Coquand–Huet Calculus of Constructions and the Other Systems in Barendregt's Cube. Technical Report. CMU-CS-88-131. CMU, Department of Computer Science and Universita di Torino, Dipartimento Matematica (1988)
13. Berghofer, S., Urban, C.: A head-to-head comparison of de Bruijn indices and names. Electron. Notes Theor. Comput. Sci. **174**(5), 53–67 (2007). https://doi.org/10.1016/j.entcs.2007.01.018
14. Bezem, M., Coquand, T., Huber, S.: A model of type theory in cubical sets. In: Matthes, R., Schubert, A. (eds.) 19th International Conference on Types for Proofs and Programs, TYPES 2013, 22–26 April 2013, Toulouse, France. LIPIcs, vol. 26, pp. 107–128. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2013). https://doi.org/10.4230/LIPIcs.TYPES.2013.107
15. Bird, R.S., Paterson, R.: De Bruijn notation as a nested datatype. J. Funct. Program. **9**(1), 77–91 (1999). https://doi.org/10.1017/S0956796899003366
16. Blanchette, J.C., Gheri, L., Popescu, A., Traytel, D.: Bindings as bounded natural functors. In: Proceedings of ACM Programming Languages 3(POPL), 2019, pp. 22:1–22:34 (2019). https://doi.org/10.1145/3290335
17. Charguéraud, A.: The locally nameless representation. J. Autom. Reason. **49**(3), 363–408 (2012). https://doi.org/10.1007/s10817-011-9225-2
18. Chlipala, A.: Parametric higher-order abstract syntax for mechanized semantics. In: Hook, J., Thiemann, P. (eds.) International Conference on Functional Programming (ICFP), 2008, pp. 143–156. ACM (2008). https://doi.org/10.1145/1411204.1411226
19. de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. Indag. Math **75**(5), 381–392 (1972). https://doi.org/10.1016/1385-7258(72)90034-0
20. Despeyroux, J., Felty, A.P., Hirschowitz, A.: Higher-order abstract syntax in Coq. In: Dezani-Ciancaglini, M., Plotkin, G.D. (eds.) Typed Lambda Calculi and Applications (TLCA) 1995, LNCS, vol. 902, pp. 124–138. Springer (1995). https://doi.org/10.1007/BFb0014049

21. Dybjer, P.: A general formulation of simultaneous inductive–recursive definitions in type theory. J. Symb. Log. **65**(2), 525–549 (2000). https://doi.org/10.2307/2586554
22. Felty, A.P., Momigliano, A.: Hybrid: a definitional two-level approach to reasoning with higher-order abstract syntax. J. Autom. Reason. **48**(1), 43–105 (2012). https://doi.org/10.1007/s10817-010-9194-x
23. Felty, A.P., Momigliano, A., Pientka, B.: The next 700 challenge problems for reasoning with higher-order abstract syntax representations—Part 2—a survey. J. Autom. Reason. **55**(4), 307–372 (2015). https://doi.org/10.1007/s10817-015-9327-3
24. Felty, A.P., Momigliano, A., Pientka, B.: An open challenge problem repository for systems supporting binders. In: Cervesato, I., Chaudhuri, K. (eds.) Proceedings Tenth International Workshop on Logical Frameworks and Meta Languages: Theory and Practice, LFMTP 2015, Berlin, Germany, 1 August 2015. EPTCS, vol. 185, pp. 18–32 (2015). https://doi.org/10.4204/EPTCS.185.2
25. Ferreira, F., Pientka, B.: Programs using syntax with first-class binders. In: Yang, H. (ed.) Programming Languages and Systems—26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, 22–29 April 2017. Proceedings. Lecture Notes in Computer Science, vol. 10201, pp. 504–529. Springer (2017). https://doi.org/10.1007/978-3-662-54434-1_19
26. Fiore, M.P., Plotkin, G.D., Turi, D.: Abstract syntax and variable binding. In: Logic in Computer Science (LICS), 1999, pp. 193–202. IEEE Computer Society (1999). https://doi.org/10.1109/LICS.1999.782615
27. Gabbay, M., Pitts, A.M.: A new approach to abstract syntax involving binders. In: Logic in Computer Science (LICS), 1999, pp. 214–224. IEEE Computer Society (1999). https://doi.org/10.1109/LICS.1999.782617
28. Gabbay, M.J., Hofmann, M.: Nominal renaming sets. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, 22–27 November 2008. Proceedings. Lecture Notes in Computer Science, vol. 5330, pp. 158–173. Springer (2008)
29. Gabbay, M.J., Mathijssen, A.: Capture-avoiding substitution as a nominal algebra. Form. Asp. Comput. **20**(4–5), 451–479 (2008). https://doi.org/10.1007/s00165-007-0056-1
30. Gabbay, M.J., Mathijssen, A.: One-and-a-halfth-order logic. J. Log. Comput. **18**(4), 521–562 (2008). https://doi.org/10.1093/logcom/exm064
31. Gheri, L., Popescu, A.: A formalized general theory of syntax with bindings: extended version. J. Autom. Reason. **64**(4), 641–675 (2020). https://doi.org/10.1007/s10817-019-09522-2
32. Goguen, H., McKinna, J.: Candidates for Substitution. Technical Report. ECS-LFCS-97-358. University of Edinburgh, School of Informatics (1997). https://www.lfcs.inf.ed.ac.uk/reports/97/ECS-LFCS-97-358/
33. Gordon, A.D., Melham, T.F.: Five axioms of alpha-conversion. In: von Wright, J., Grundy, J., Harrison, J. (eds.) Theorem Proving in Higher Order Logics, 9th International Conference, TPHOLs'96, Turku, Finland, 26–30 August 1996, Proceedings. Lecture Notes in Computer Science, vol. 1125, pp. 173–190. Springer (1996). https://doi.org/10.1007/BFb0105404
34. Gunter, E.L., Osborn, C.J., Popescu, A.: Theory support for weak higher order abstract syntax in Isabelle/HOL. In: Cheney, J., Felty, A.P. (eds.) Logical Frameworks and Meta-languages: Theory and Practice (LFMTP), 2009, pp. 12–20. ACM (2009). https://doi.org/10.1145/1577824.1577827
35. Harper, R., Honsell, F., Plotkin, G.D.: A framework for defining logics. In: Logic in Computer Science (LICS), 1987, pp. 194–204. IEEE Computer Society (1987). https://doi.org/10.1145/138027.138060
36. Hofmann, M.: Semantical analysis of higher-order abstract syntax. In: Logic in Computer Science (LICS), 1999, pp. 204–213. IEEE Computer Society (1999). https://doi.org/10.1109/LICS.1999.782616
37. Institute for Advanced Study: The Univalent Foundations Program: Homotopy Type Theory. Univalent Foundations of Mathematics. Institute for Advanced Study (2013). https://homotopytypetheory.org/book
38. Johnstone, P.T.: Quotients of decidable objects in a topos. Math. Proc. Camb. Philos. Soc. **93**, 409–419 (1983). https://doi.org/10.1017/S0305004100060734
39. Kaiser, J., Schäfer, S., Stark, K.: Binder aware recursion over well-scoped de Bruijn syntax. In: Andronick, J., Felty, A.P. (eds.) Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, 8–9 January 2018. pp. 293–306. ACM (2018). https://doi.org/10.1145/3167098
40. Kammüller, F., Wenzel, M., Paulson, L.C.: Locales—a sectioning concept for Isabelle. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin-Mohring, C., Théry, L. (eds.) Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September 1999, Proceedings. Lecture Notes in Computer Science, vol. 1690, pp. 149–166. Springer (1999). https://doi.org/10.1007/3-540-48256-3_11
41. McKinna, J., Pollack, R.: Pure type systems formalized. In: Bezem, M., Groote, J.F. (eds.) Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA

'93, Utrecht, The Netherlands, 16–18 March 1993, Proceedings. Lecture Notes in Computer Science, vol. 664, pp. 289–305. Springer (1993). https://doi.org/10.1007/BFb0037113
42. McKinna, J., Pollack, R.: Some lambda calculus and type theory formalized. J. Autom. Reason. **23**(3–4), 373–409 (1999)
43. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL—A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Berlin (2002)
44. Norrish, M.: Recursive function definition for types with binders. In: Slind, K., Bunker, A., Gopalakrishnan, G. (eds.) Theorem Proving in Higher Order Logics (TPHOLs), 2004, LNCS, vol. 3223, pp. 241–256. Springer (2004). https://doi.org/10.1007/978-3-540-30142-4_18
45. Paulson, L.C.: The foundation of a generic theorem prover. J. Autom. Reason. **5**(3), 363–397 (1989). https://doi.org/10.1007/BF00248324
46. Paulson, L.C., Blanchette, J.C.: Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, 9 October 2011. EPiC Series in Computing, vol. 2, pp. 1–11. EasyChair (2010). https://doi.org/10.29007/36dt
47. Pfenning, F., Elliott, C.: Higher-order abstract syntax. In: Wexelblat, R.L. (ed.) Programming Language Design and Implementation (PLDI), 1988, pp. 199–208. ACM (1988). https://doi.org/10.1145/53990.54010
48. Pfenning, F., Schürmann, C.: System description: Twelf—a meta-logical framework for deductive systems. In: Ganzinger, H. (ed.) Conference on Automated Deduction (CADE), 1999, LNCS, vol. 1632, pp. 202–206. Springer (1999). https://doi.org/10.1007/3-540-48660-7_14
49. Pientka, B.: Beluga: programming with dependent types, contextual data, and contexts. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) Functional and Logic Programming (FLOPS), 2010, LNCS, vol. 6009, pp. 1–12. Springer (2010). https://doi.org/10.1007/978-3-642-12251-4_1
50. Pitts, A.: Locally nameless sets. In: Proceedings of ACM Programming Languages 7(POPL), 2023 (2023)
51. Pitts, A.M.: Nominal logic, a first order theory of names and binding. Inf. Comput. **186**(2), 165–193 (2003). https://doi.org/10.1016/S0890-5401(03)00138-X
52. Pitts, A.M.: Alpha-structural recursion and induction. J. ACM **53**(3), 459–506 (2006). https://doi.org/10.1145/1147954.1147961
53. Pitts, A.M.: Nominal Sets: Names and Symmetry in Computer Science. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge (2013)
54. Pitts, A.M.: Nominal presentation of cubical sets models of type theory. In: Herbelin, H., Letouzey, P., Sozeau, M. (eds.) 20th International Conference on Types for Proofs and Programs (TYPES 2014). Leibniz International Proceedings in Informatics (LIPIcs), vol. 39, pp. 202–220. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl (2015). http://drops.dagstuhl.de/opus/volltexte/2015/5498
55. Pollack, R., Sato, M., Ricciotti, W.: A canonical locally named representation of binding. J. Autom. Reason. **49**(2), 185–207 (2012)
56. Popescu, A.: Contributions to the theory of syntax with bindings and to process algebra. PhD Thesis, University of Illinois at Urbana-Champaign (2010). https://www.andreipopescu.uk/pdf/thesisUIUC.pdf
57. Popescu, A.: Rensets and renaming-based recursion for syntax with bindings. In: Blanchette, J., Kovacs, L., Pattinson, D. (eds.) International Joint Conference on Automated Reasoning (IJCAR), 2022. Lecture Notes in Computer Science, vol. 13385, pp. 618–639. Springer (2022)
58. Popescu, A.: Renaming-enriched sets (rensets) and renaming-based recursion. In: Archives of Formal Proofs 2023 (2023). https://www.isa-afp.org/entries/Rensets.html
59. Popescu, A., Gunter, E.L.: Recursion principles for syntax with bindings and substitution. In: Chakravarty, M.M.T., Hu, Z., Danvy, O. (eds.) Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, 19–21 September 2011, pp. 346–358. ACM (2011). https://doi.org/10.1145/2034773.2034819
60. Popescu, A., Gunter, E.L., Osborn, C.J.: Strong normalization for system F by HOAS on top of FOAS. In: Logic in Computer Science (LICS), 2010, pp. 31–40. IEEE Computer Society (2010). https://doi.org/10.1109/LICS.2010.48
61. Popescu, A., Roşu, G.: Term-generic logic. Theor. Comput. Sci. **577**, 1–24 (2015)
62. Schäfer, S., Tebbi, T., Smolka, G.: Autosubst: reasoning with de Bruijn terms and parallel substitutions. In: Urban, C., Zhang, X. (eds.) Interactive Theorem Proving (ITP), 2015, LNCS, vol. 9236, pp. 359–374. Springer (2015). https://doi.org/10.1007/978-3-319-22102-1_24
63. Schürmann, C., Despeyroux, J., Pfenning, F.: Primitive recursion for higher-order abstract syntax. Theor. Comput. Sci. **266**(1–2), 1–57 (2001). https://doi.org/10.1016/S0304-3975(00)00418-7
64. Stark, K.: Mechanising syntax with binders in Coq. PhD Thesis, Saarland University, Saarbrücken (2020). https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/28822

65. Staton, S.: Name-Passing Process Calculi: Operational Models and Structural Operational Semantics. Technical Report. UCAM-CL-TR-688. University of Cambridge, Computer Laboratory (2007). https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-688.pdf
66. Stoughton, A.: Substitution revisited. Theor. Comput. Sci. **59**, 317–325 (1988). https://doi.org/10.1016/0304-3975(88)90149-1
67. Sun, Y.: An algebraic generalization of Frege structures—binding algebras. Theor. Comput. Sci. **211**(1–2), 189–232 (1999)
68. Urban, C.: Nominal techniques in Isabelle/HOL. J. Autom. Reason. **40**(4), 327–356 (2008). https://doi.org/10.1007/s10817-008-9097-2
69. Urban, C., Berghofer, S.: A recursion combinator for nominal datatypes implemented in Isabelle/HOL. In: Furbach, U., Shankar, N. (eds.) International Joint Conference on Automated Reasoning (IJCAR), 2006, LNCS, vol. 4130, pp. 498–512. Springer (2006). https://doi.org/10.1007/11814771_41
70. Urban, C., Berghofer, S., Norrish, M.: Barendregt's variable convention in rule inductions. In: Pfenning, F. (ed.) Conference on Automated Deduction (CADE), 2007, LNCS, vol. 4603, pp. 35–50. Springer (2007). https://doi.org/10.1007/978-3-540-73595-3_4
71. Urban, C., Kaliszyk, C.: General bindings and alpha-equivalence in Nominal Isabelle. Log. Methods Comput. Sci. (2012). https://doi.org/10.2168/LMCS-8(2:14)2012
72. Urban, C., Tasson, C.: Nominal techniques in Isabelle/HOL. In: Nieuwenhuis, R. (ed.) Conference on Automated Deduction (CADE), 2005. LNCS, vol. 3632, pp. 38–53. Springer (2005). https://doi.org/10.1007/11532231_4