Journal of Logical and Algebraic Methods in Programming 91 (2017) 17–32

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Completeness and incompleteness in nominal Kleene algebra

Dexter Kozen [a,*], Konstantinos Mamouras [b], Alexandra Silva [c]

[a] Computer Science Department, Cornell University, Ithaca, NY 14853-7501, USA
[b] CIS Department, University of Pennsylvania, Philadelphia, PA 19104-6309, USA
[c] Department of Computer Science, University College London, London WC1E 6BT, UK

ARTICLE INFO

Article history:
Received 26 January 2016
Received in revised form 9 June 2017
Accepted 11 June 2017
Available online 16 June 2017

Keywords:
Kleene algebra
Nominal sets
Programming logic

ABSTRACT

Gabbay and Ciancia (2011) presented a nominal extension of Kleene algebra as a framework for trace semantics with statically scoped allocation of resources, along with a semantics consisting of nominal languages. They also provided an axiomatization that captures the behavior of the scoping operator and its interaction with the Kleene algebra operators and proved soundness over nominal languages. In this paper, we show that the axioms proposed by Gabbay and Ciancia are not complete over the semantic interpretation they propose. We then identify a slightly wider class of language models over which they are sound and complete.

1. Introduction

Nominal sets are a convenient framework for handling name generation and binding. They were introduced by Gabbay and Pitts [1] as a mathematical model of name binding and α-conversion.

Nominal extensions of classical automata theory have recently been explored [2], motivated by the increasing need for tools for languages over infinite alphabets. These play a role in various areas, including XML document processing, cryptography, and verification. An XML document can be seen as a tree with labels from the (infinite) set of all unicode strings that can appear as attribute values. In cryptography, infinite alphabets are used as *nonces*, names used only once in cryptographic communications to prevent replay attacks. In software verification, infinite alphabets are used for references, objects, pointers, and function parameters.

In this paper, we focus on axiomatizations of the regular languages and how they can be lifted in the presence of a binding operator ν and an infinite alphabet of names. This work builds on the recent work of Gabbay and Ciancia [3], who presented a nominal extension of Kleene algebra (KA), called *nominal Kleene algebra* (NKA), as a framework for trace semantics with statically scoped allocation of resources. Gabbay and Ciancia [3] also presented an interpretation *NL* for NKA over *nominal languages* and provided a set of six axioms (Definition 2.8) that capture the behavior of the scoping operator ν and its interaction with the usual Kleene algebra operators. They showed that these axioms, in conjunction with the KA axioms (Definition 2.1), are sound over *NL*, but left open the question of completeness. In this paper we address this problem.

We first show (Theorem 4.1) that the axioms are not complete for the language model *NL* proposed by Gabbay and Ciancia. This is due to the presence of what Gabbay and Ciancia call *non-maximal planes*. The issue is rather technical, but

* Corresponding author.
E-mail addresses: kozen@cs.cornell.edu (D. Kozen), mamouras@seas.upenn.edu (K. Mamouras), alexandra.silva@gmail.com (A. Silva).
URLs: http://www.cs.cornell.edu/~kozen (D. Kozen), http://www.seas.upenn.edu/~mamouras (K. Mamouras), http://www.alexandrasilva.org (A. Silva).

http://dx.doi.org/10.1016/j.jlamp.2017.06.002
2352-2208/

$$
\begin{array}{ll}
x+(y+z)=(x+y)+z & x(yz)=(xy)z \\
x+y=y+x & x0=0x=0 \\
x+0=x+x=x & 1x=x1=x \\
x(y+z)=xy+xz & (x+y)z=xz+yz \\
\\
1+xx^*\le x^* & 1+x^*x\le x^* \\
y+xz\le z \;\Rightarrow\; x^*y\le z & y+zx\le z \;\Rightarrow\; yx^*\le z
\end{array}
$$

Fig. 1. Axioms of Kleene algebra (KA).

we explain it in detail in §3.1 and show exactly why it causes completeness to fail. We then present in §3.2 our alternative language model *AL* that is a mild modification of *NL* and develop some basic properties of the model. We also introduce in §3.3 a variety of sound interpretations in which the scoping operator is interpreted as a summation operator over a fixed set. The axioms are not complete over these models either, but for rather uninteresting reasons. However, these models attest to the versatility of NKA.

Our main result is completeness of the axioms over *AL* (Theorem 4.2). Completeness is achieved by first transforming each expression to an equivalent expression for which only the usual Kleene algebra axioms (Definition 2.1) are needed. The steps of the transformation make use of the KA axioms along with axioms proposed by Gabbay and Ciancia (Definition 2.8) for the scoping operator. The proof is quite long but is broken into four steps: *exposing bound variables*, *scope configuration*, *canonical choice of bound variables*, and *semilattice identities*. In the last step, we make use of a technique of [4] that exploits the fact that the Boolean algebra generated by finitely many regular sets consists of regular sets and is atomic. Hence, expressions can be written as sums of atoms.

The paper is organized as follows. In §2 we recall basic material on nominal sets, Kleene algebra (KA), and nominal Kleene algebra (NKA) of Gabbay and Ciancia [3]. In §3, we discuss various interpretations: the original language model *NL* proposed in [3], our own alternative language model *AL*, and the summation models. We give a precise description of the difference between *NL* and *AL*. In §4, we present our main results on incompleteness and completeness (Theorems 4.1 and 4.2, respectively). In §5 we present concluding remarks and directions for future work.

2. Background

In this section we review basic background material on Kleene algebra (KA), nominal sets, and the nominal extension of KA (NKA) of Gabbay and Ciancia [3]. For a more thorough introduction, the reader is referred to [5,6] for nominal sets, to [7] for Kleene (co)algebra, and to [3] for NKA.

2.1. Kleene algebra (KA)

Kleene algebra is the algebra of regular expressions. Regular expressions are normally interpreted as regular sets of strings, but there are other useful interpretations: binary relation models used in programming language semantics, the $(\min,+)$ algebra used in shortest path algorithms, models consisting of convex sets used in computational geometry, and many others.

Definition 2.1 *(Kleene algebra (KA)).* A *Kleene algebra* is any structure $(K,+,\cdot,^*,0,1)$ where K is a set, $+$ and $\cdot$ are binary operations on K, * is a unary operation on K, and 0 and 1 are constants, satisfying the axioms listed in Fig. 1, where we define $x\le y$ iff $x+y=y$. The top block of eleven axioms (those not involving *) are succinctly stated by saying that the structure is an idempotent semiring under $+,\cdot,0$, and 1. The term *idempotent* refers to the axiom $x+x=x$. Due to this axiom, the ordering relation $\le$ is a partial order. The remaining four axioms involving * together say that x^*y is the $\le$-least z such that $y+xz\le z$ and yx^* is the $\le$-least z such that $y+zx\le z$.

2.2. Nominal sets

Nominal sets originated in the work of Fraenkel in 1922 and were originally used to prove the independence of the axiom of choice and other axioms. They were introduced in computer science by Gabbay and Pitts [1] as a formalism for modeling name binding in quantificational logic and the λ-calculus. Since then there has been a substantial amount of research on nominal sets in a wide variety of fields related to logic in computer science. Recent work includes the development of new programming languages [8,9] and their use in learning of automata [10].

In a nutshell, nominal sets are sets closed under the action of a certain group of symmetries $G_{\mathbb{A}}$ (defined below) and satisfying a certain finite-support condition. Nominal sets may be infinite, but the closure under symmetries makes them finitely representable and tractable for algorithms.

Definition 2.2 *(Group action and G-set).* A *group action* of a group G on a set X is a map $G \times X \to X$, written as juxtaposition, such that $\pi(\rho x) = (\pi\rho)x$ and $1x = x$ for all $\pi, \rho \in G$ and $x \in X$. Here 1 is the unit of group G.

A *G-set* is a set X equipped with a group action $G \times X \to X$. A function $f : X \to Y$ between G-sets is called *equivariant* if it commutes with the group action; that is, for all $x \in X$ and $\pi \in G$, $f(\pi x) = \pi(f(x))$. Viewing π as a map $x \mapsto \pi x$, we can write $f \circ \pi = \pi \circ f$. The G-sets and equivariant functions form a category.

Definition 2.3 *(*$\mathsf{fix}\, x$ *and* $\mathsf{Fix}\, A$*).* Let X be a G-set. For $x \in X$ and $A \subseteq X$, define the subgroups

$$\mathsf{fix}\, x = \{\pi \in G \mid \pi x = x\} \qquad \mathsf{Fix}\, A = \bigcap_{x \in A} \mathsf{fix}\, x = \{\pi \in G \mid \forall x \in A\ \pi x = x\}.$$

If X is a G-set, then so is its powerset, with $\pi A = \{\pi x \mid x \in A\}$ for $A \subseteq X$. In general, $\mathsf{fix}\, A = \{\pi \in G \mid \pi A = A\}$ and $\mathsf{Fix}\, A$ are different: $\mathsf{fix}\, A$ fixes A setwise, whereas $\mathsf{Fix}\, A$ fixes A pointwise.

Let $\mathbb{A}$ be a fixed countably infinite set of *atoms*. The permutations of $\mathbb{A}$, that is, the bijective functions $\pi : \mathbb{A} \to \mathbb{A}$, form a group in which the identity permutation is the unit element, inverse is functional inverse, and multiplication is function composition.

Definition 2.4 *(Nominal sets).* Let $G_{\mathbb{A}}$ be the group of all *finite* permutations of $\mathbb{A}$, that is, permutations generated by transpositions $(a\ b)$. The group $G_{\mathbb{A}}$ acts on $\mathbb{A}$ in the obvious way, making $\mathbb{A}$ into a $G_{\mathbb{A}}$-set. If X is another $G_{\mathbb{A}}$-set, we say that $A \subseteq \mathbb{A}$ *supports* $x \in X$ if $\mathsf{Fix}\, A \subseteq \mathsf{fix}\, x$; in other words, any finite permutation that fixes A pointwise also fixes x. An element $x \in X$ has *finite support* if there is a finite set $A \subseteq \mathbb{A}$ that supports x. A *nominal set* is a $G_{\mathbb{A}}$-set X such that every element of X has finite support.

It can be shown that if $A, B \subseteq \mathbb{A}$ and $A \cup B \neq \mathbb{A}$, then $\mathsf{Fix}(A \cap B)$ is the smallest subgroup of $G_{\mathbb{A}}$ containing both $\mathsf{Fix}\, A$ and $\mathsf{Fix}\, B$. Thus if A and B are finite and support x, then so does $A \cap B$. It follows that if x is finitely supported, there is a unique $\subseteq$-minimal set that supports it.

Definition 2.5 *(Support, freshness).* Let X be a nominal set and $x \in X$. The *support* of x, denoted $\mathsf{supp}\, x$, is the unique setwise minimal finite subset of $\mathbb{A}$ that supports x. We write $a\#x$ and say a is *fresh for* x if $a \notin \mathsf{supp}\, x$.

The standard example of a nominal set is the set of λ-terms over variables $\mathbb{A}$. The set of free variables of a λ-term is its support. Any permutation of the variables that fixes the free variables is considered to fix the term; this captures the idea of α-equivalence in the framework of nominal sets. A variable is fresh for a λ-term if it has no free occurrence in that term.

One can show that A supports x iff πA supports πx. In particular, $\mathsf{supp}\, \pi x = \pi\, \mathsf{supp}\, x$, thus the function supp is equivariant. Also, for $x \in X$, $\mathsf{Fix}\, \mathsf{supp}\, x \subseteq \mathsf{fix}\, x \subseteq \mathsf{fix}\, \mathsf{supp}\, x$, and both inclusions can be strict.

2.3. Nominal Kleene algebra (NKA)

Definition 2.6 *(Syntax of NKA).* The abstract syntax of nominal Kleene algebra (NKA) expressions over an alphabet Σ of primitive letters is defined by the following grammar:

$$e ::= a \in \Sigma \mid e + e \mid ee \mid e^* \mid 0 \mid 1 \mid \nu a.e.$$

Thus expressions are just KA expressions with the addition of the binding construct $\nu a.e$. The scope of the binding νa in $\nu a.e$ is e. By convention, we take the precedence of the binding operator νa to be lower than product and star but higher than sum; thus in products, scopes extend as far to the right as possible. For example, $\nu a.ab\ \nu b.ba$ should be read as $\nu a.(ab\ \nu b.(ba))$ and not $(\nu a.ab)(\nu b.ba)$. The set of NKA expressions over Σ is denoted Exp_Σ.

For a finite set $A = \{a_1, \ldots, a_n\} \subseteq \mathbb{A}$, we will use the notation $\nu A.e$ as a shorthand for $\nu a_1.\nu a_2.\cdots\nu a_n.e$.

Definition 2.7 *(ν-strings and Σ^ν).* A *ν-string* over an alphabet Σ is an expression with no occurrence of $+$, *, or 0, and no occurrence of 1 except to denote the null string, in which case we use ε instead:

$$x ::= a \in \Sigma \mid xx \mid \varepsilon \mid \nu a.x.$$

Two ν-strings that are equivalent modulo associativity of multiplication are considered equal. The set of ν-strings over Σ is denoted Σ^ν.

The *free variables* $\mathsf{FV}(e)$ of an expression or ν-string e are defined inductively as usual. We write $e[a/x]$ for the result of substituting a for variable x in e.

The following six axioms were proposed by Gabbay and Ciancia [3] to describe how the binding operator ν interacts with the KA operators.

Definition 2.8 *(Nominal axioms [3]).*

(N1) $\nu a.(d+e) = \nu a.d + \nu a.e$
(N2) $\nu a.\nu b.e = \nu b.\nu a.e$
(N3) $a\#e \Rightarrow \nu a.e = e$
(N4) $a\#e \Rightarrow \nu b.e = \nu a.(a\ b)e$
(N5) $a\#e \Rightarrow (\nu a.d)e = \nu a.de$
(N6) $a\#e \Rightarrow e(\nu a.d) = \nu a.ed$.

Intuitively, (N1) says that variable binding commutes with $+$; (N2) says that binding of two variables can be done in either order; (N3) says that a binding is useless if nothing is bound; (N4) says that a fresh name can be swapped in for a bound variable (this is essentially α-conversion); and (N5) and (N6) say that a scope may be extended on the right or left provided no free variable is captured.

3. Models

Definition 3.1 *(Nominal Kleene algebra (NKA)).* A *nominal Kleene algebra* (NKA) over atoms $\mathbb{A}$ is a structure $(K, +, \cdot, ^*, 0, 1, \nu)$ with binding operation $\nu : \mathbb{A} \times K \to K$ such that

- K is a nominal set over atoms $\mathbb{A}$; that is, there is a group action $G_{\mathbb{A}} \times K \to K$ such that all elements of K have finite support;
- the KA operations and ν are equivariant in the sense that

$$\pi(x+y) = \pi x + \pi y \qquad \pi(xy) = (\pi x)(\pi y) \qquad \pi 0 = 0$$
$$\pi(x^*) = (\pi x)^* \qquad \pi(\nu a.e) = \nu(\pi a).(\pi e) \qquad \pi 1 = 1$$

  for all $\pi \in G_{\mathbb{A}}$; that is, the action of every $\pi \in G_{\mathbb{A}}$ is an automorphism of K; and
- all the KA and nominal axioms (Definitions 2.1 and 2.8, respectively) are satisfied.

3.1. Nominal language model

Now we describe an interpretation $NL : \mathsf{Exp}_{\mathbb{A}} \to \mathcal{P}(\mathbb{A}^*)$ that interprets NKA expressions over $\mathbb{A}$ as certain subsets of $\mathbb{A}^*$, the set of finite-length strings over $\mathbb{A}$. This is the nominal language model introduced in [3]. Our definition is equivalent to that of [3] but broken into a two-step process involving an intermediate interpretation I over ν-strings. As noted in [3], care must be taken when defining the product of languages to avoid capture.

First we give an intermediate interpretation $I : \mathsf{Exp}_{\mathbb{A}} \to \mathcal{P}(\mathbb{A}^\nu)$ of expressions as sets of ν-strings over $\mathbb{A}$.

Definition 3.2. Define $I : \mathsf{Exp}_{\mathbb{A}} \to \mathcal{P}(\mathbb{A}^\nu)$ to be the homomorphic interpretation in which the regular operators $+$, $\cdot$, *, 0, and 1 have their usual set-theoretic interpretations, and

$$I(\nu a.e) = \{\nu a.x \mid x \in I(e)\} \qquad I(a) = \{a\}.$$

We thus maintain the scoping of ν-subexpressions in the ν-strings.

For example,

$$I(\nu a.a) = \{\nu a.a\}$$
$$I(\nu a.\nu b.(a+b)) = \{\nu a.\nu b.a, \nu a.\nu b.b\}$$
$$I(\nu a.(\nu b.ab)(a+b)) = \{\nu a.(\nu b.ab)a, \nu a.(\nu b.ab)b\}$$
$$I(\nu a.(ab)^*) = \{\nu a.\varepsilon, \nu a.ab, \nu a.abab, \nu a.ababab, \ldots\}$$
$$I((\nu a.ab)^*) = \{\varepsilon, \nu a.ab, (\nu a.ab)(\nu a.ab), (\nu a.ab)(\nu a.ab)(\nu a.ab), \ldots\}.$$

Finally, we describe the map $NL : \mathbb{A}^\nu \to \mathcal{P}(\mathbb{A}^*)$ on ν-strings and $NL : \mathsf{Exp}_{\mathbb{A}} \to \mathcal{P}(\mathbb{A}^*)$ on NKA expressions. Given a ν-string x, first α-convert so that all bindings in x are distinct and different from all free variables in x, then delete all binding operators νa to obtain a string $x' \in \mathbb{A}^*$. For example, $(\nu a.ab)(\nu a.ab)(\nu a.ab)' = abcbdb$. Here we have α-converted to obtain $(\nu a.ab)(\nu c.cb)(\nu d.db)$, then deleted the binding operators to obtain $abcbdb$. The choice of variables in the α-conversion does not matter as long as they are distinct and different from the free variables. Then for each ν-string x and expression e, define

$$NL(x) = \{\pi x' \mid \pi \in \mathsf{Fix}\,\mathsf{FV}(x)\} \qquad NL(e) = \bigcup_{x \in I(e)} NL(x), \tag{3.1}$$

where x' is the string obtained from x as described above.

The set $NL(x)$ is the plane $x'\circlearrowleft_{\mathsf{FV}(x)}$ in the terminology and notation of [3].[1] Thus we let the bound variables range simultaneously over all possible values in $\mathbb{A}$ they could take on, as long as they remain distinct and different from the free variables, and we accumulate all strings obtained in this way. For example,

$$\begin{aligned} NL(\nu a.a) &= \mathbb{A} \\ NL(\nu a.\nu b.(a+b)) &= \mathbb{A} \\ NL(\nu a.(\nu b.ab)b) &= \{acb \mid a, c \in \mathbb{A},\ a, c, b \text{ distinct}\} \\ NL(\nu a.(ab)^*) &= \{(ab)^n \mid n \geq 0,\ a \in \mathbb{A},\ a \neq b\} \\ NL((\nu a.ab)^*) &= \{a_1 b a_2 b \cdots a_n b \mid n \geq 0,\ a_i \in \mathbb{A},\ \text{all } a_i \neq a_j,\ a_i \neq b\}. \end{aligned}$$

As mentioned, the fresh variables used in the α-conversion do not matter, thus for any $y \in NL(x)$,

$$NL(x) = \{\pi\, y \mid \pi \in \mathsf{Fix}\,\mathsf{FV}(x)\}. \tag{3.2}$$

For ν-strings $x, y \in \mathbb{A}^\nu$, write $x \equiv y$ if x and y are equivalent modulo the nominal axioms (Definition 2.8). The following lemma says that the nominal axioms alone are sound and complete for equivalence between ν-strings in the nominal language model.

Lemma 3.3. *For $x, y \in \mathbb{A}^\nu$, $x \equiv y$ if and only if $NL(x) = NL(y)$.*

Proof. Soundness (the left-to-right implication) holds because each nominal axiom preserves NL, as is not difficult to check. For completeness (the right-to-left implication), suppose $NL(x) = NL(y)$. We must have $\mathsf{FV}(x) = \mathsf{FV}(y)$, because if $a \in \mathsf{FV}(x) - \mathsf{FV}(y)$, then $NL(y)$ would contain a string with no occurrence of a, whereas all strings in $NL(x)$ contain an occurrence of a. Now α-convert x and y using (N4) so that all bound variables are distinct and different from the free variables, and extend all scopes to the entire string using (N5) and (N6), so that $x \equiv \nu A.x'$ and $y \equiv \nu B.y'$ for some $x', y' \in \mathbb{A}^*$. By (3.2), $x' = \pi\, y'$ for some $\pi \in \mathsf{Fix}\,\mathsf{FV}(x) = \mathsf{Fix}\,\mathsf{FV}(y)$, so $x \equiv \pi\, y$, and $\pi\, y \equiv y$ by α-conversion. □

Lemma 3.4. *For any $x \in \mathbb{A}^*$ and $A, B \subseteq \mathsf{FV}(x)$,*

$$A \subseteq B \Leftrightarrow NL(\nu A.x) \subseteq NL(\nu B.x)$$

(in the notation of [3], $A \subseteq B \Leftrightarrow x\circlearrowleft_{B'} \subseteq x\circlearrowleft_{A'}$, where $A' = \mathsf{FV}(x) - A$ and $B' = \mathsf{FV}(x) - B$).

Proof. If $A \subseteq B$, then $B' \subseteq A'$, so $\mathsf{Fix}\,A' \subseteq \mathsf{Fix}\,B'$. Then

$$NL(\nu A.x) = \{\pi\, x \mid \pi \in \mathsf{Fix}\,A'\} \subseteq \{\pi\, x \mid \pi \in \mathsf{Fix}\,B'\} = NL(\nu B.x).$$

Conversely, if $a \in A - B$, then $x[b/a] \in NL(\nu A.x) - NL(\nu B.x)$, where b is any element of $\mathbb{A} - \mathsf{FV}(x)$. □

Lemma 3.5. *Let $y \in NL(e)$ and $A \subseteq \mathsf{FV}(y)$ maximal such that $NL(\nu A.y) \subseteq NL(e)$ (in the notation of [3], this is $y\circlearrowleft_{A'} \propto NL(e)$, where $A' = \mathsf{FV}(y) - A$). Then $\nu A.y \in I(e)$, and $\nu A.y$ is the unique ν-string up to nominal equivalence for which this is true.*

Proof. Let $x_1, \ldots, x_n \in I(e)$ be all ν-strings such that $y \in NL(x_i)$. There are only finitely many of these. Then

$$NL(\nu A.y) \subseteq NL(x_1) \cup \cdots \cup NL(x_n) \subseteq NL(e).$$

Using the nominal axioms (Definition 2.8), we can move the quantification in each x_i to the front of the string and α-convert so that the quantifier-free part is y. This is possible because $y \in NL(x_i)$. Thus we can assume without loss of generality that each $x_i = \nu A_i.y$ for some $A_i \subseteq \mathsf{FV}(y)$.

Let $z \in NL(\nu A.y)$ such that $(\mathsf{FV}(z) - \mathsf{FV}(\nu A.y)) \cap \mathsf{FV}(\nu A_i.y) = \varnothing$ for $1 \leq i \leq n$. Since

$$NL(\nu A.y) \subseteq NL(x_1) \cup \cdots \cup NL(x_n) = NL(\nu A_1.y) \cup \cdots \cup NL(\nu A_n.y),$$

we must have $z \in NL(\nu A_i.y)$ for some i. But then $\mathsf{FV}(\nu A.y), \mathsf{FV}(\nu A_i.y) \subseteq \mathsf{FV}(z)$ and $\mathsf{FV}(\nu A_i.y) \subseteq \mathsf{FV}(\nu A.y)$ by choice of z, therefore $A \subseteq A_i$. Since A was maximal, $A = A_i$. □

Lemma 3.5 gives the essential content of [3, Theorem 3.16]. This is important for us because it says that the set $NL(e)$ uniquely determines the maximal elements of $I(e)$ up to nominal equivalence (Lemma 3.7 below).

[1] We do not present the construction of [3], as it is rather involved; however, we will occasionally provide translations for the benefit of readers who are familiar with that work.

Definition 3.6. Let $\hat{I}(e) = \{x \in I(e) \mid NL(x) \text{ is maximal in } NL(e)\}$.

Lemma 3.7. *$NL(e_1) = NL(e_2)$ if and only if $\hat{I}(e_1) = \hat{I}(e_2)$ modulo the nominal axioms (Definition 2.8).*

Proof. Suppose $NL(e_1) = NL(e_2)$. By Lemma 3.5, each $y \in NL(e_1)$ is contained in a unique maximal $NL(\nu A.y)$, and $\nu A.y \in \hat{I}(e_1)$. As $NL(e_1) = NL(e_2)$, these planes are also contained in $NL(e_2)$. Symmetrically, the maximal planes of $NL(e_2)$ are contained in $NL(e_1)$. Since the two sets contain the same set of maximal planes, they must be equal, therefore $\hat{I}(e_1) = \hat{I}(e_2)$ modulo the nominal axioms.

For the reverse implication, note that

$$NL(e) = \bigcup_{x \in I(e)} NL(x) = \bigcup_{x \in \hat{I}(e)} NL(x)$$

by the fact that every plane of e is contained in a maximal one. Then

$$NL(e_1) = \bigcup_{x \in \hat{I}(e_1)} NL(x) = \bigcup_{x \in \hat{I}(e_2)} NL(x) = NL(e_2). \qquad \square$$

3.2. Alternative nominal language model

In this section we present a new language-theoretic interpretation, denoted *AL*, which is an alternative to the interpretation *NL* of Gabbay and Ciancia [3] described in §3.1. The two interpretations are quite close, except that *AL* distinguishes between free and bound names.

Let Σ and $\mathbb{A}$ be countably infinite disjoint sets. For this section, we let the letters $a, b, c, \ldots$ range over $\mathbb{A}$, $x, y, z, \ldots$ over Σ, and $u, v, w, \ldots$ over $(\Sigma \cup \mathbb{A})^*$. Binding with ν is only over Σ.

A *language* is a subset $A \subseteq (\Sigma \cup \mathbb{A})^*$ such that $\pi A = A$ for all $\pi \in G_{\mathbb{A}}$. The set of languages is denoted $\mathcal{L}$.

Definition 3.8. The operations of nominal KA (Definition 2.6) are defined on $\mathcal{L}$ as follows:

$$A + B = A \cup B \qquad AB = \{uv \mid u \in A,\ v \in B,\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\}$$
$$A^* = \bigcup_n A^n \qquad \nu x.A = \{w[a/x] \mid w \in A,\ a \in \mathbb{A} - \mathsf{FV}(w)\},\ x \in \Sigma$$
$$0 = \varnothing \qquad 1 = \{\varepsilon\}.$$

In words, $\nu x.A$ is the set of strings obtained by substituting $a \in \mathbb{A}$ for x in strings $w \in A$ for all possible choices of $a \in \mathbb{A}$ and $w \in A$, subject to the condition that a does not already occur in w. Set concatenation AB is like the usual set concatenation operator of formal language theory in which words from A are concatenated with words from B, except here the two words may not have any letters of $\mathbb{A}$ in common.

Lemma 3.9. *The set $\mathcal{L}$ is closed under the set-theoretic operations of Definition 3.8.*

Proof. For sum, $\pi(\bigcup_n A_n) = \bigcup_n \pi A_n = \bigcup_n A_n$. For product,

$$\begin{aligned}
\pi(AB) &= \{\pi(uv) \mid u \in A,\ v \in B,\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\
&= \{(\pi u)(\pi v) \mid u \in A,\ v \in B,\ \mathsf{FV}(\pi u) \cap \mathsf{FV}(\pi v) \cap \pi\mathbb{A} = \varnothing\} \\
&= \{uv \mid u \in \pi A,\ v \in \pi B,\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\
&= (\pi A)(\pi B) = AB.
\end{aligned}$$

The case of A^* follows from the previous two cases. The cases of 0 and 1 are trivial. Finally, for $\nu x.A$, we have

$$\begin{aligned}
\pi(\nu x.A) &= \{\pi(w[a/x]) \mid w \in A,\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\
&= \{(\pi w)[\pi a/x] \mid w \in A,\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\
&= \{w[a/x] \mid \pi^{-1} w \in A,\ \pi^{-1} a \in \mathbb{A} - \mathsf{FV}(\pi^{-1} w)\} \\
&= \{w[a/x] \mid w \in \pi A,\ a \in \pi\mathbb{A} - \pi\mathsf{FV}(\pi^{-1} w)\} \\
&= \{w[a/x] \mid w \in A,\ a \in \mathbb{A} - \mathsf{FV}(w)\} = \nu x.A. \qquad \square
\end{aligned}$$

We can interpret nominal KA expressions as languages in $\mathcal{L}$. The interpretation map $AL : \mathsf{Exp}_\Sigma \to \mathcal{L}$ is the unique homomorphism with respect to the language operations of Definition 3.8 such that $AL(x) = \{x\}$. Note that in this context, atoms $a \in \mathbb{A}$ do not appear in expressions or ν-strings.

Theorem 3.10. *The nominal axioms (Definition 2.8) hold under the interpretation AL.*

Proof. The proofs of $AL(\nu x.(d+e)) = AL(\nu x.d) \cup AL(\nu x.e)$ and $AL(\nu x.\nu y.e) = AL(\nu y.\nu x.e)$ are straightforward. For the axiom $\nu x.(d+e) = \nu x.d + \nu x.e$,

$$\begin{aligned} AL(\nu x.(d+e)) &= \{w[a/x] \mid w \in AL(d+e),\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= \{w[a/x] \mid w \in AL(d) \cup AL(e),\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= \{w[a/x] \mid w \in AL(d),\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &\qquad \cup \{w[a/x] \mid w \in AL(e),\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= AL(\nu x.d) \cup AL(\nu x.e). \end{aligned}$$

For the axiom $\nu x.\nu y.e = \nu y.\nu x.e$,

$$\begin{aligned} AL(\nu x.\nu y.e) &= \{w[a/x] \mid w \in AL(\nu y.e),\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= \{w[a/x] \mid w \in \{u[b/y] \mid u \in AL(e),\ b \in \mathbb{A} - \mathsf{FV}(u)\},\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= \{u[b/y][a/x] \mid u \in AL(e),\ b \in \mathbb{A} - \mathsf{FV}(u),\ a \in \mathbb{A} - \mathsf{FV}(u[b/y])\} \\ &= \{u[a/x][b/y] \mid u \in AL(e),\ a \in \mathbb{A} - \mathsf{FV}(u),\ b \in \mathbb{A} - \mathsf{FV}(u[a/x])\} \\ &= AL(\nu y.\nu x.e). \end{aligned}$$

For the remaining axioms, assume $x \notin \mathsf{FV}(e)$.

$$\begin{aligned} AL(\nu x.e) = \nu x.AL(e) &= \{w[a/x] \mid w \in AL(e),\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= \{w \mid w \in AL(e),\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= AL(e). \end{aligned}$$

The next-to-last equation holds since $\mathsf{FV}(w) \cap \Sigma \subseteq \mathsf{FV}(e)$, therefore $x \notin \mathsf{FV}(w)$, so $w[a/x] = w$.

$$\begin{aligned} AL(\nu y.e) &= \nu y.AL(e) \\ &= \{w[a/y] \mid w \in AL(e),\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= \{((x\ y)w)[a/x] \mid (x\ y)w \in AL((x\ y)e),\ a \in \mathbb{A} - \mathsf{FV}((x\ y)w)\} \\ &= \{w[a/x] \mid w \in AL((x\ y)e),\ a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= AL(\nu x.(x\ y)e). \end{aligned}$$

$$\begin{aligned} AL((\nu x.d)e) &= \{uv \mid u \in AL(\nu x.d),\ v \in AL(e),\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\ &= \{uv \mid u \in \nu x.AL(d),\ v \in AL(e),\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\ &= \{uv \mid u \in \{w[a/x] \mid w \in AL(d),\ a \in \mathbb{A} - \mathsf{FV}(w)\},\ v \in AL(e), \\ &\qquad \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\ &= \{w[a/x]v \mid w \in AL(d),\ a \in \mathbb{A} - \mathsf{FV}(w),\ v \in AL(e), \\ &\qquad \mathsf{FV}(w[a/x]) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\ &= \{w[a/x]v \mid w \in AL(d),\ a \in \mathbb{A} - \mathsf{FV}(wv),\ v \in AL(e), \\ &\qquad \mathsf{FV}(w) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \qquad (3.3) \\ &= \{u[a/x]v \mid u \in AL(d),\ v \in AL(e),\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing, \\ &\qquad a \in \mathbb{A} - \mathsf{FV}(uv)\} \\ &= \{(uv)[a/x] \mid u \in AL(d),\ v \in AL(e),\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing, \\ &\qquad a \in \mathbb{A} - \mathsf{FV}(uv)\} \\ &= \{w[a/x] \mid w \in \{uv \mid u \in AL(d),\ v \in AL(e),\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\}, \\ &\qquad a \in \mathbb{A} - \mathsf{FV}(w)\} \\ &= \{w[a/x] \mid w \in AL(de),\ a \in \mathbb{A} - \mathsf{FV}(w)\} = \nu x.AL(de) = AL(\nu x.de). \end{aligned}$$

All steps are straightforward except (3.3), which requires an argument. We consider two cases: either $x \in \mathsf{FV}(w)$ or $x \notin \mathsf{FV}(w)$. In the former case, we argue that

$$\begin{aligned} &a \in \mathbb{A} - \mathsf{FV}(w) \text{ and } \mathsf{FV}(w[a/x]) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing \\ &\quad \Leftrightarrow a \in \mathbb{A} - \mathsf{FV}(wv) \text{ and } \mathsf{FV}(w) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing. \end{aligned}$$

For the left-to-right implication, since $x \in \mathsf{FV}(w)$, we have $a \in \mathsf{FV}(w[a/x]) \cap \mathbb{A}$, and since $\mathsf{FV}(w[a/x]) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing$, it must be that $a \notin \mathsf{FV}(v)$, therefore $a \in \mathbb{A} - (\mathsf{FV}(w) \cup \mathsf{FV}(v)) = \mathbb{A} - \mathsf{FV}(wv)$. Also, $\mathsf{FV}(w) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing$, since $\mathsf{FV}(w[a/x]) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing$ and $\mathsf{FV}(w) \cap \mathbb{A} \subseteq \mathsf{FV}(w[a/x]) \cap \mathbb{A}$.

For the right-to-left implication, since

$$a \in \mathbb{A} - \mathsf{FV}(wv) = \mathbb{A} - (\mathsf{FV}(w) \cup \mathsf{FV}(v)),$$

we have $a \in \mathbb{A} - \mathsf{FV}(w)$ and $a \in \mathbb{A} - \mathsf{FV}(v)$, therefore

$$\begin{aligned} \mathsf{FV}(w[a/x]) \cap \mathsf{FV}(v) \cap \mathbb{A} &= (\mathsf{FV}(w) \cup \{a\}) \cap \mathsf{FV}(v) \cap \mathbb{A} \\ &= (\mathsf{FV}(w) \cap \mathsf{FV}(v) \cap \mathbb{A}) \cup (\{a\} \cap \mathsf{FV}(v) \cap \mathbb{A}) \\ &= \varnothing \cup \varnothing = \varnothing. \end{aligned}$$

In the latter case ($x \notin \mathsf{FV}(w)$), the equation (3.3) reduces to

$$\begin{aligned} &\{wv \mid w \in AL(d),\ a \in \mathbb{A} - \mathsf{FV}(w),\ v \in AL(e),\ \mathsf{FV}(w) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\ &\quad = \{wv \mid w \in AL(d),\ a \in \mathbb{A} - \mathsf{FV}(wv),\ v \in AL(e),\ \mathsf{FV}(w) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\}, \end{aligned}$$

which is clearly true, as a is irrelevant.

Similarly, $AL(e(\nu x.d)) = AL(\nu x.ed)$. □

The following lemma is analogous to Lemma 3.3. It implies that on ν-strings, *AL* and *NL* are equivalent in deductive power (under the obvious modification that expressions are over Σ and not $\mathbb{A}$). However, for *AL*, we have a third equivalent condition (iii) that does not hold for *NL*.

Lemma 3.11. *For $u, v \in \Sigma^\nu$, the following are equivalent:*

(i) $u \equiv v$
(ii) $AL(u) = AL(v)$
(iii) $AL(u) \cap AL(v) \neq \varnothing$.

Proof. Certainly (i) implies (ii) by soundness (Theorem 3.10), and (ii) implies (iii) since both $AL(u)$ and $AL(v)$ are nonempty. To show (iii) implies (i), as in the proof of Lemma 3.3, we can assume without loss of generality that $u = \nu A.u'$ and $v = \nu B.v'$ for some $u', v' \in \Sigma^*$ with no useless binders. By (iii), u' and v' must contain the same free and bound variables (up to α-conversion) and in the same order, otherwise there could be no common string obtained by substituting distinct elements of $\mathbb{A}$ for the bound variables. We can thus rearrange the binders and α-convert using (N2) and (N4) to show equivalence. □

We can also define $I : \mathsf{Exp}_\Sigma \to \Sigma^\nu$ and $\hat{I} : \mathsf{Exp}_\Sigma \to \Sigma^\nu$ exactly as in §3.1 for the nominal language model, again with the modification that expressions are over Σ and not $\mathbb{A}$.

Lemma 3.12. $AL(e) = \bigcup_{w \in I(e)} AL(w)$.

Proof. This can be proved by a straightforward induction on the structure of e. We argue the case of products and binders explicitly.

$$\begin{aligned} AL(e_1e_2) &= \{uv \mid u \in AL(e_1),\ v \in AL(e_2),\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\ &= \{uv \mid u \in \bigcup_{p \in I(e_1)} AL(p),\ v \in \bigcup_{q \in I(e_2)} AL(q),\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\ &= \bigcup_{\substack{p \in I(e_1) \\ q \in I(e_2)}} \{uv \mid u \in AL(p),\ v \in AL(q),\ \mathsf{FV}(u) \cap \mathsf{FV}(v) \cap \mathbb{A} = \varnothing\} \\ &= \bigcup_{\substack{p \in I(e_1) \\ q \in I(e_2)}} AL(pq) = \bigcup_{r \in I(e_1e_2)} AL(r). \end{aligned}$$

$$\begin{aligned}
AL(\nu x.e) &= \nu x.AL(e)\\
&= \{w[a/x] \mid w \in AL(e),\ a \in \mathbb{A} - \mathsf{FV}(w)\}\\
&= \{w[a/x] \mid w \in \bigcup_{p \in I(e)} AL(p),\ a \in \mathbb{A} - \mathsf{FV}(w)\}\\
&= \bigcup_{p \in I(e)} \{w[a/x] \mid w \in AL(p),\ a \in \mathbb{A} - \mathsf{FV}(w)\}\\
&= \bigcup_{p \in I(e)} \nu x.AL(p) = \bigcup_{p \in I(e)} AL(\nu x.p) = \bigcup_{w \in I(\nu x.e)} AL(w). \qquad \square
\end{aligned}$$

Lemma 3.13. *Every plane $AL(\nu A.w)$ in $AL(e)$ is maximal; that is, $I(e) = \hat{I}(e)$.*

Proof. Replace each $x \in A$ in w with a distinct element of $\mathbb{A}$ to get w'. Then $AL(\nu A.w) = \{\pi\, w' \mid \pi \in G_{\mathbb{A}}\}$. This is maximal, as all finite permutations of $\mathbb{A}$ are allowed. $\square$

Lemma 3.13 characterizes the key difference between the nominal language model *NL* of [3] described in §3.1 and the alternative nominal language model *AL* of this section. It explains why the axioms are complete for the alternative model but not for the model of §3.1. In the model of §3.1, there are non-maximal planes, and these are "hidden" by the maximal planes, whereas this cannot happen in the alternative model, as all planes are maximal.

3.3. Summation models

We end this section with a description of several other interesting models in which ν is interpreted as some form of summation operator: a summation model over the free KA, a summation model over languages, a summation model over an arbitrary KA, and an evaluation model. The axioms are sound over these models, but not complete. We include these models as a testament to the versatility of NKA.

3.3.1. Summation model over the free KA

It is sound to interpret νx as a summation operator $\sum_{a \in F} e[a/x]$. Let K be the free KA on generators X (regular expressions over X modulo the KA axioms (Definition 2.1)) and let $F \subseteq K$. For $x \in X$, interpret

$$\nu x.e = \sum_{a \in F} e[a/x] \qquad x\#e \Leftrightarrow x \notin \mathsf{FV}(e),$$

where $e[a/x]$ is the expression obtained by substituting a for x in e. This can be interpreted in any KA in which the sums exist; and in any KA when F is finite. This is a sound interpretation, as all the nominal axioms are satisfied:

$$\sum_a (d+e)[a/x] = \sum_a d[a/x] + \sum_a e[a/x] \qquad x \notin \mathsf{FV}(e) \Rightarrow \sum_a e[a/x] = e$$

$$\sum_a \sum_b e[b/y][a/x] = \sum_b \sum_a e[a/x][b/y] \qquad x \notin \mathsf{FV}(e) \Rightarrow (\sum_a d[a/x])e = \sum_a (de)[a/x]$$

$$x \notin \mathsf{FV}(e) \Rightarrow \sum_a e[a/y] = \sum_a e[x/y][a/x] \qquad x \notin \mathsf{FV}(e) \Rightarrow e(\sum_a d[a/x]) = \sum_a (ed)[a/x].$$

3.3.2. Language summation model

In particular, let $\mathbb{A}$ be a set of letters, finite or infinite. We wish to interpret expressions as subsets of $\mathbb{A}^*$ by a map $L : \mathsf{Exp}_{\mathbb{A}} \to \mathcal{P}(\mathbb{A}^*)$. Let the regular operators have their usual set-theoretic interpretations, and let

$$L(\nu x.e) = \bigcup_{a \in \mathbb{A}} L(e[a/x]) \qquad L(a) = \{a\},\ a \in \mathbb{A}.$$

3.3.3. Summation model over an arbitrary KA K

More generally, let K be an arbitrary KA and let $K[X]$ be the set of polynomials over indeterminates X with coefficients in K. This is the direct sum (coproduct) of K with the free KA on generators X. Let $F \subseteq K$ be finite. Let $e \mapsto e[a/x]$ be the evaluation morphism that for $x \in X$ evaluates $e \in K[X]$ at $x = a$. We can interpret

$$\nu x.e = \sum_{a \in F} e[a/x] \qquad x\#e \Leftrightarrow e \in K[X - \{x\}] \Leftrightarrow e[0/x] = e.$$

This is also a sound interpretation:

$$\sum_a (d+e)[a/x] = \sum_a d[a/x] + \sum_a e[a/x] \qquad e[0/x] = e \Rightarrow \sum_a e[a/x] = e$$

$$\sum_a \sum_b e[b/y][a/x] = \sum_b \sum_a e[a/x][b/y] \qquad e[0/x] = e \Rightarrow \sum_a e[a/y] = \sum_a e[x/y][a/x]$$

$$e[0/x] = e \Rightarrow (\sum_a d[a/x])e = \sum_a (de)[a/x] \qquad e[0/x] = e \Rightarrow e(\sum_a d[a/x]) = \sum_a (ed)[a/x].$$

3.3.4. Evaluation model

In fact the set F in the model of §3.3.3 can be a singleton $\{a\}$; in this case, $\nu x.e$ is just evaluation $e \mapsto e[a/x]$.

$$(d+e)[a/x] = d[a/x] + e[a/x] \qquad e[a/y][a/x] = e[a/x][a/y]$$

$$e[0/x] = e \Rightarrow e[a/x] = e \qquad e[0/x] = e \Rightarrow e[a/y] = e[x/y][a/x]$$

$$e[0/x] = e \Rightarrow (d[a/x])e = (de)[a/x] \qquad e[0/x] = e \Rightarrow e(d[a/x]) = (ed)[a/x].$$

4. Completeness and incompleteness

In this section we prove our main theorems.

Theorem 4.1 *(Incompleteness over NL). The language interpretation NL of Gabbay and Ciancia (§3.1) satisfies equations that are not consequences of the axioms of NKA (Definitions 2.1 and 2.8).*

Proof. As is easily verified, the inequality $a \le \nu a.a$ holds in *NL*, since

$$NL(a) = \{a\} \subseteq \mathbb{A} = NL(\nu a.a).$$

However, this equation does not hold in any of the summation models of §3.3, nor under the interpretation *AL*. Since the axioms are sound for these models, all consequences of the axioms hold in them, thus $a \le \nu a.a$ is not a consequence of the axioms. □

Henceforth, we let $a, b, \ldots$ as well as $x, y, \ldots$ range over Σ. It is also the case that none of the summation models provide a free nominal KA, as $\nu a.aa \le \nu a.\nu b.ab$ holds in all the summation models but not in our language model *AL*. However, the axioms of NKA are complete for *NL* if one restricts to closed terms, as the closed terms of *NL* and *AL* coincide.

Theorem 4.2 *(Completeness over AL and nominal Kleene algebras). The axioms of nominal Kleene algebra (Definitions 2.1 and 2.8) are sound and complete for the equational theory of nominal Kleene algebras (Definition 3.1) and for the equational theory of the alternative language interpretation AL of §3.2.*

Theorem 4.2 says that if two NKA expressions e_1 and e_2 are equivalent in the alternative language interpretation of §3.2 in the sense that $AL(e_1) = AL(e_2)$, then e_1 and e_2 are provably equivalent in equational logic from the axioms of NKA (Definitions 2.1 and 2.8). The remainder of §4 is devoted to the proof of this result.

To prove Theorem 4.2, we show that every expression can be put into a certain canonical form that will allow us to apply the KA axioms to prove equivalence. The construction consists of four main steps, expounded in the next four subsections: *exposing bound variables* (§4.1), *scope configuration* (§4.2), *canonical choice of bound variables* (§4.3), and *determining semilattice identities* (§4.4). Each step will involve a construction that is justified by the axioms.

For the purposes of exposition, we write $\underset{a}{\underset{a}{(}} e \underset{a}{\underset{a}{)}}$ instead of $\nu a.e$ so that it is easier to see the scope boundaries. In this notation, the nominal axioms take the following form:

$$\nu a.(d+e) = \nu a.d + \nu a.e \qquad \underset{a}{\underset{a}{(}} d+e \underset{a}{\underset{a}{)}} = \underset{a}{\underset{a}{(}} d \underset{a}{\underset{a}{)}} + \underset{a}{\underset{a}{(}} e \underset{a}{\underset{a}{)}} \tag{4.1}$$

$$\nu a.\nu b.e = \nu b.\nu a.e \qquad \underset{a}{\underset{b}{(}}\underset{b}{\underset{a}{(}} e \underset{b}{\underset{a}{)}}\underset{a}{\underset{b}{)}} = \underset{b}{\underset{a}{(}}\underset{a}{\underset{b}{(}} e \underset{a}{\underset{b}{)}}\underset{b}{\underset{a}{)}} \tag{4.2}$$

$$a\#e \Rightarrow \nu a.e = e \qquad a\#e \Rightarrow \underset{a}{\underset{a}{(}} e \underset{a}{\underset{a}{)}} = e \tag{4.3}$$

$$a\#e \Rightarrow \nu b.e = \nu a.(a\ b)e \qquad a\#e \Rightarrow \underset{b}{\underset{b}{(}} e \underset{b}{\underset{b}{)}} = \underset{a}{(}(a\ b)e\underset{a}{)} \tag{4.4}$$

$$a\#e \Rightarrow (\nu a.d)e = \nu a.de \qquad a\#e \Rightarrow \underset{a}{\underset{a}{(}} d \underset{a}{\underset{a}{)}} e = \underset{a}{\underset{a}{(}} de \underset{a}{\underset{a}{)}} \tag{4.5}$$

$$a\#e \Rightarrow e(\nu a.d) = \nu a.ed \qquad a\#e \Rightarrow e\,\underset{a}{\underset{a}{(}} d \underset{a}{\underset{a}{)}} = \underset{a}{\underset{a}{(}} ed \underset{a}{\underset{a}{)}}. \tag{4.6}$$

We remark that writing scope boundaries of ν-expressions as letters $\underset{a}{(}$ and $\underset{a}{)}$ is merely a notational convenience. Although it appears to allow us to violate the invariant that starred expressions and ν-expressions are mutually well-nested, in reality this is not an issue, as all our transformations are justified by the axioms, which maintain this invariant.

4.1. Exposing bound variables

Definition 4.3 *(ν^*-strings).* Define a *ν^*-string* to be a string of

- letters a,
- well-nested scope delimiters $\underset{a}{(}$ and $\underset{a}{)}$, and
- starred expressions e^* whose bodies e are (inductively) sums of ν^*-strings.

Definition 4.4 *(Exposure).* We say that the bound variables of a ν^*-string are *exposed* if

(i) the first and last occurrence of each bound variable occur at the top level in the scope of their binding operator,[2] and
(ii) the bound variables of all ν^*-strings in the bodies of starred subexpressions are (inductively) exposed.

For example, a typical ν^*-string is $\underset{a}{(}\underset{b}{(}abb\underset{a}{(}ab\underset{a}{)}\underset{a}{(}ab\underset{a}{)}+b\underset{b}{(}ba\underset{b}{)}\underset{b}{)}^*ba\underset{b}{)}\underset{a}{)}$. The bound variables are exposed in this expression because the first and last occurrences of a and b occur at the top level. Inside the starred subexpression, the bound variables in the two ν^*-strings are exposed because there are no starred subexpressions.

The purpose of exposing bound variables is to create a barrier for scope delimiters $\underset{a}{(}$ and $\underset{a}{)}$ as they move inward. This will become clear in §4.2.

Lemma 4.5. *Every expression e can be transformed to an equivalent sum of ν^*-strings e' whose bound variables are exposed. Moreover, the value of the interpretation $I : \mathsf{Exp}_{\mathbb{A}} \to \mathcal{P}(\mathbb{A}^\nu)$ defined in §3.1 is invariant under the transformation.*

Proof. It is straightforward to see how to use the nominal axiom (4.1) in the left-to-right direction and the distributivity and 0 and 1 laws of Kleene algebra to write every expression as a sum of ν^*-strings.

Exposing the bound variables is a little more difficult. It may appear at first glance that one can simply unwind e^* as $1+e+ee^*e$ and then unwind the starred subexpressions of e inductively, but this is not enough. For example,

$$\begin{aligned}(a+b)^* &= 1+a+b+(a+b)(a+b)^*(a+b)\\ &= 1+a+b+a(a+b)^*a+a(a+b)^*b+b(a+b)^*a+b(a+b)^*b,\end{aligned}$$

and the subexpression $a(a+b)^*a$ does not satisfy (i). The following more complicated expression is needed:

$$\begin{aligned}(a+b)^* = {} & 1+a+b+aa^*a+bb^*b+ab+ba && (4.7)\\ & +aa^*ab+aa^*ba^*a+baa^*a+abb^*b+bb^*ab^*b+bb^*ba && (4.8)\\ & +aa^*abb^*b+aa^*b(a+b)^*ab^*b+aa^*b(a+b)^*ba^*a && (4.9)\\ & +bb^*a(a+b)^*ab^*b+bb^*a(a+b)^*ba^*a+bb^*baa^*a. && (4.10)\end{aligned}$$

Line (4.7) covers strings containing no a's or no b's or one of each. Line (4.8) covers strings containing one a and two or more b's or one b and two or more a's. Lines (4.9) and (4.10) cover strings containing at least two a's and at least two b's.

For the general construction, we first argue the case of $(a_1+\cdots+a_n)^*$. Write down all strings containing either zero, one, or two occurrences of each letter. For each such string, insert a starred subexpression in each gap between adjacent letters. The body of the starred expression inserted into a gap will be the sum of all letters a such that the gap falls between two occurrences of a.

For example, the second term of (4.9) is obtained from the string $abab$. There are three gaps, into which we insert the indicated starred expressions:

$$\begin{array}{ccccc} a & & b & & a & & b \\ & \uparrow & & \uparrow & & \uparrow & \\ & a^* & & (a+b)^* & & b^* & \end{array}$$

In the first gap we inserted a^* because the gap falls between two occurrences of a but not between two occurrences of b. In the second gap we inserted $(a+b)^*$ because the gap falls between two occurrences of a and two occurrences of b.

[2] "Top level" means not inside a starred subexpression. Inside a starred expression e^*, "top level" means not inside a starred subexpression of e.

This construction covers all strings whose first and last occurrences of each letter occur in the order specified by the original string before the insertion. If a letter occurs twice before the insertion, then after the insertion those two occurrences are the first and last, and they occur at the top level. If a letter occurs once before the insertion, then that is the only occurrence after the insertion, and it is at the top level. If a letter does not occur at all before the insertion, then it does not occur after.

For the general case e^*, we first perform the construction inductively on all starred subexpressions of e, writing $e^* = (e_1 + \cdots + e_n)^*$ where each top-level ν^*-string e_i satisfies conditions (i) and (ii) of Definition 4.4. Now take the sum $f = f_1 + \cdots + f_k$ constructed above for $(a_1 + \cdots + a_n)^*$ (in the example (4.7)–(4.10), $n = 2$ and $k = 19$) and substitute e_i for a_i in f to obtain $f[e_i/a_i]$. We claim that this expression is of the desired form. Any ν^*-string generated by $(e_1 + \cdots + e_n)^*$ is generated by some $e_{i_1} e_{i_2} \cdots e_{i_m}$, which is a substitution instance of $a = a_{i_1} a_{i_2} \cdots a_{i_m}$, which in turn is generated by some f_i. For any letter c, let e_{i_j} be the leftmost expression in $e_{i_1} \cdots e_{i_m}$ in which c occurs. Then the first occurrence of a_{i_j} in f_i occurs at the top level, the first occurrence of c in e_{i_j} occurs at the top level, and c does not occur in $e_{i_1} \cdots e_{i_{j-1}}$. Therefore that occurrence of c is the first occurrence in $f_i[e_i/a_i]$ and occurs at the top level.

The value of the interpretation $I : \mathsf{Exp}_{\mathbb{A}} \to \mathcal{P}(\mathbb{A}^\nu)$ defined in §3.1 is preserved by the transformation, that is, $I(e) = I(e')$, because the only operations that were performed were justified by the KA axioms and (4.1), all of which preserve I. Note that the axioms (4.2)–(4.6) do not preserve I, but we have not used those in the transformation. □

4.2. Scope configuration

For this part of the construction, we first α-convert using (4.4) to make all bound variables distinct and different from any free variable. This is commonly known as the *Barendregt variable convention*.

Now we transform each ν^*-string to ensure that every top-level left delimiter $(_a$ occurs immediately to the left of an occurrence of a that it binds:

$$\cdots (_a \cdots (_b \cdots (_c \cdots)_c \cdots)_b \cdots)_a \cdots \tag{4.11}$$

That occurrence is at the top level due to the preprocessing step of §4.1. We do this without changing the order of any occurrences of variables in the string, but we may change the order of quantification.

Starting at the left end of the string, scan right, looking for top-level left delimiters. For all top-level left delimiters that we see, push them to the right as long as we do not encounter a variable bound by any of them. Stop when such a variable is encountered. For example,

$$\cdots (_a \cdots (_b \cdots (_c \cdots b \cdots)_c \cdots)_b \cdots)_a \cdots \;\Rightarrow\; \cdots (_a (_b (_c \cdots)_c \cdots)_b \cdots)_a \cdots$$

Here we are using the nominal axiom (4.6) in the right-to-left direction to skip over letters and starred expressions. If such a variable is encountered, it will be at the top level because of the preprocessing step of §4.1.

In this example, we must keep the $(_b$ to the left of that occurrence of b, but we wish to move the $(_a$ and $(_c$ past the b. The c can be moved in using (4.6), but to move the a in, we must exchange the order of quantification of a and b. To do this, we push the corresponding right delimiter of b up to the right delimiter of a using the nominal axiom (4.5) in the left-to-right direction.

$$\cdots (_a (_b (_c \cdots)_c \cdots)_b \cdots)_a \cdots \;\Rightarrow\; \cdots (_a (_b (_c \cdots)_c \cdots)_b)_a \cdots$$

This is always possible, as there is no free occurrence of b to the right of the $)_b$ due to the Barendregt variable convention. Now we can exchange the order of quantification using the nominal axiom (4.2).

$$\cdots (_a (_b (_c \cdots)_c \cdots)_b)_a \cdots \;\Rightarrow\; \cdots (_b (_a (_c \cdots)_c \cdots)_a)_b \cdots$$

This allows us to move the $(_a$ and $(_c$ in past the b and continue.

$$\cdots (_b (_a (_c \cdots)_c \cdots)_a)_b \cdots \;\Rightarrow\; \cdots (_b (_a (_c \cdots)_c \cdots)_a)_b \cdots$$

When looking for the first occurrence of a variable bound to a left delimiter, perhaps no such occurrence is encountered before seeing a right delimiter. In this case there is nothing bound, so we can just forget the binding altogether.

$$\cdots (_a (_b (_c (_c \cdots)_c \cdots b \cdots)_b \cdots)_a \cdots \;\Rightarrow\; \cdots (_a (_b \cdots b \cdots)_b \cdots)_a \cdots$$

This uses the nominal axiom (4.3).

If there exists an occurrence of a bound to $\underset{a}{(}\cdots\underset{a}{)}$, then the leftmost one occurs at the top level due to the construction of §4.1. Thus, when we are done, any remaining left delimiters $\underset{a}{(}$ in the string occur immediately to the left of an occurrence of a that is bound to that delimiter, as illustrated in (4.11).

Now we finish up the construction by moving the right delimiters to the left as far as possible using (4.5) in the right-to-left direction without exchanging order of quantification. Because of the preprocessing step of §4.1, the rightmost occurrence of any variable quantified at the top level occurs at the top level. Thus every right delimiter $\underset{a}{)}$ occurs either immediately to the right of an occurrence of a bound to that delimiter or immediately to the right of another right delimiter $\underset{b}{)}$ with smaller scope.

At this point we have transformed the expression so that every ν^*-string satisfies the following properties:

(i) every ν-subformula is of the form $\nu a.ae$; that is, the leftmost symbol of every scope is a variable bound by that scope; and
(ii) the rightmost boundary of every scope is as far to the left as possible, subject to (i).

The position of the scope delimiters is canonical, because scopes are as small as possible: the left delimiters are as far to the right as they can possibly be, and the right delimiters are as far to the left as they can possibly be given the positions of the left delimiters.

Let us denote by $'' : \mathsf{Exp}_{\mathbb{A}} \to \mathsf{Exp}_{\mathbb{A}}$ the transformation of this subsection (§4.2) combined with the transformation of the previous subsection (§4.1) that exposes bound variables. The construction is reflected in the semantics, as expressed by the following lemma.

Lemma 4.6. *Let $I : \mathsf{Exp}_{\mathbb{A}} \to \mathcal{P}(\mathbb{A}^\nu)$ be the intermediate interpretation of Definition 3.2. Then*

$$I(e'') = \{w'' \mid w \in I(e)\}.$$

Proof. We have already argued in the proof of Lemma 4.5 that $I(e) = I(e')$, where $'$ is the transformation of §4.1 that exposes bound variables. Each of the transformation steps in the construction of this subsection commutes with the interpretation I, as can be shown formally by induction on the structure of the expression, thus

$$I(e'') = \{w'' \mid w \in I(e')\} = \{w'' \mid w \in I(e)\}. \qquad \square$$

Lemma 4.7. *Any two equivalent NKA expressions, after transforming them by $''$, generate the same set of ν-strings up to renaming of bound variables. That is, if $AL(e_1) = AL(e_2)$, then $I(e_1'')$ and $I(e_2'')$ are equal up to renaming of bound variables.*

Proof. We have $AL(e_i) = \bigcup_{w \in I(e_i)} AL(w)$ for $i \in \{1, 2\}$ by Lemma 3.12 and $AL(e_i) = AL(e_i'')$ for $i \in \{1, 2\}$ by Theorem 3.10. Putting these together, if $AL(e_1) = AL(e_2)$, then

$$\bigcup_{w \in I(e_1'')} AL(w) = \bigcup_{w \in I(e_2'')} AL(w).$$

By Lemma 3.11, inequivalent ν-strings have disjoint interpretations under AL, therefore $I(e_1'')$ and $I(e_2'')$ must be equal modulo $\equiv$; that is, for all $u \in I(e_1'')$, there exists $v \in I(e_2'')$ such that $u \equiv v$ and vice versa.

But we also have $I(e_i'') = \{w'' \mid w \in I(e_i)\}$ by Lemma 4.6, therefore all elements of $I(e_1'')$ and $I(e_2'')$ are in $''$-normal form, which is unique. Thus if $u \equiv v$, then u and v are α-equivalent. Therefore $I(e_1'') = I(e_2'')$ up to α-equivalence. $\square$

4.3. Canonical choice of bound variables

Now we would like to transform the expression so that the bound variables are chosen in a canonical way. This will ensure that if two expressions are equivalent, then they generate the same ν-strings, not just up to renaming of bound variables, but absolutely. This part of the construction will thus relax the Barendregt variable convention, so that variables can be bound more than once and can occur both bound and free in a string.

Choose a set of variables disjoint from the free variables of the expression and order them in some arbitrary but fixed order $a_0, a_1, \ldots$. Moving through the expression from left to right, maintain a stack of variable names corresponding to the scopes we are currently in. When a left scope delimiter $\underset{a}{(}$ is encountered, and we are inside the scope of n ν-formulas, the variables $a_0, \ldots, a_{n-1}$ will be on the stack. We rename the bound variable a to a_n using the nominal axiom (4.4) for α-conversion and push a_n onto the stack. When a right scope delimiter is encountered, we pop the stack. This construction guarantees that every ν-string generated by the expression satisfies:

- For every symbol in the string, if the symbol occurs in the scope of n nested ν-expressions, then those expressions bind variables $a_0, \ldots, a_{n-1}$ in that order from outermost to innermost scope.

It follows that two semantically equivalent expressions so transformed generate exactly the same set of ν-strings. We have shown

Lemma 4.8. *Let $'''$ be the composition of the constructions of §4.1–4.3. If $AL(e_1) = AL(e_2)$, then $I(e_1''') = I(e_2''')$.*

4.4. Determining semilattice identities

After transforming expressions by the constructions $'''$ of §4.1–4.3, we know by Lemma 4.8 that if e_1 and e_2 are equivalent, then they generate the same sets of ν-strings; that is, $I(e_1''') = I(e_2''')$, where I is the map defined in Definition (3.2). Now we wish to show that any two such expressions can be proved equivalent using the KA and nominal axioms in conjunction with the following congruence rule for ν-formulas:

$$\frac{e_1 = e_2}{\nu a.e_1 = \nu a.e_2}. \tag{4.12}$$

In order to do this, there is one more issue that must be resolved. To introduce this issue, let us first assume for simplicity that e_1 and e_2 are of ν-depth one; that is, they only contain bindings of one variable a. There may be several subexpressions in e_1 and e_2 of the form $\nu a.d$, but all with the same variable a. We will relax this restriction later.

Any substring of the form $\nu a.w$ of a ν-string generated by e must be generated by a subexpression of the form $\nu a.d$. However, there may be several different subexpressions of this form, and the string $\nu a.w$ could be generated by more than one of them. In general, the sets of ν-strings generated by the ν-subexpressions could satisfy various semilattice identities, and we may have to know these identities in order to prove equivalence.

For example, consider the two expressions $c_1 + c_2$ and $d_1 + d_2 + d_3$, where

$$\begin{aligned} c_1 &= \nu a.a(aa)^* & d_1 &= \nu a.a(aaa)^* \\ c_2 &= \nu a.aa(aa)^* & d_2 &= \nu a.aa(aaa)^* \\ & & d_3 &= \nu a.aaa(aaa)^*. \end{aligned} \tag{4.13}$$

The expression c_i generates all nonnull ν-strings with $i \bmod 2$ a's and d_i generates all nonnull ν-strings with $i \bmod 3$ a's. Both $c_1 + c_2$ and $d_1 + d_2 + d_3$ generate all nonnull ν-strings of a's, but in different ways. If $c_1 + c_2$ occurs in e_1 and $d_1 + d_2 + d_3$ occurs in e_2, we would have to know that they are equivalent to prove the equivalence of e_1 and e_2.

To determine all semilattice identities such as $c_1 + c_2 = d_1 + d_2 + d_3$ that hold among the ν-subexpressions, we express every ν-subexpression in e_1 or e_2 as a sum of atoms of the Boolean algebra of sets of ν-strings generated by these ν-subexpressions. An *atom* of a Boolean algebra is a minimal nonzero element. In a finite Boolean algebra, every element can be written as a disjunction of atoms. The family of regular sets over a fixed finite alphabet forms a Boolean algebra under the usual set-theoretic Boolean operations, as the regular sets are closed under union, intersection, and complement. Any finite collection of regular sets generates a finite subalgebra. The atoms are the minimal nonzero elements of this subalgebra.

In the example above, the atoms of the generated Boolean algebra are the sets of strings generated by $b_i = \nu a.a^i(a^6)^*$, $1 \le i \le 6$ (b_i generates strings with $i \bmod 6$ a's). Rewriting the expressions (4.13) as sums of atoms, we would obtain

$$\begin{aligned} c_1 &= b_1 + b_3 + b_5 & d_1 &= b_1 + b_4 \\ c_2 &= b_2 + b_4 + b_6 & d_2 &= b_2 + b_5 \\ & & d_3 &= b_3 + b_6. \end{aligned}$$

The equivalences are provable in pure KA plus the nominal axiom (4.1). Then $c_1 + c_2$ and $d_1 + d_2 + d_3$ become

$$\begin{aligned} c_1 + c_2 &= (b_1 + b_3 + b_5) + (b_2 + b_4 + b_6) \\ d_1 + d_2 + d_3 &= (b_1 + b_4) + (b_2 + b_5) + (b_3 + b_6), \end{aligned}$$

which are clearly equivalent.

Now we observe that any ν-string $\nu a.w$ generated by e_1 or e_2 is generated by exactly one atom. Moreover, if $\nu a.f$ is an atom and $\nu a.w \in I(\nu a.f)$, and if $\nu a.w$ is generated by $\nu a.f$ in the context $u(\nu a.w)v \in I(\nu a.e_1)$, then for any other $\nu a.z \in I(\nu a.f)$, we have $u(\nu a.z)v \in I(\nu a.e_1)$ as well. This says that we may treat $\nu a.f$ as atomic. In fact, once we have determined the atoms, if we like we may replace each atom $\nu a.f$ by a single letter $a_{\nu a.f}$ in e_1 and e_2, and the resulting expressions are equivalent, therefore provable. Then a proof of the two expressions with the letters $a_{\nu a.f}$ can be transformed back to a proof with the atoms $\nu a.f$ by simply substituting $\nu a.f$ for $a_{\nu a.f}$. However, note that it is not necessary to do the actual substitution; we can carry out the same proof on the original expressions with the $\nu a.f$.

For example, suppose $e_1 = c_1^*(c_2c_1^*)^*$ and $e_2 = (d_1 + d_2 + d_3)^*$, for which we have $I(e_1) = I(e_2)$. We expand the subexpressions c_1, c_2 and d_1, d_2, d_3 using the atoms $b_1, \ldots, b_6$ to obtain

$$e_1 = (b_1 + b_3 + b_5)^*((b_2 + b_4 + b_6)(b_1 + b_3 + b_5)^*)^*$$
$$e_2 = ((b_1 + b_4) + (b_2 + b_5) + (b_3 + b_6))^*.$$

Because of the pairwise disjointness of the atoms, these expressions must be equivalent as pure KA expressions, regarding the b_i as atomic letters, therefore can be proved equivalent in pure KA.

For expressions of ν-depth greater than one, we perform the above construction inductively, innermost scopes first. We use the KA axioms and the semilattice identities on depth-n ν-subexpressions to determine the semilattice identities on depth-$(n-1)$ ν-subexpressions, then use the nominal axiom (4.1) and the rule (4.12) to prepare these semilattice identities for use on the next level.

This completes the proof of Theorem 4.2.

5. Conclusion

We have presented results on completeness and incompleteness of nominal Kleene algebra as introduced by Gabbay and Ciancia [3]. There are various directions for future work.

The normalization procedure presented in this paper yields a decision procedure that, although effective, is likely to be prohibitively expensive in practice due to combinatorial explosions in the preprocessing step of §4.1 and in the intersection of regular expressions in §4.4. In a companion paper [11], we have explored the coalgebraic theory of nominal Kleene algebra with the aim of developing a more efficient coalgebraic decision procedure, which would be of particular interest for the applications mentioned in the introduction. Coalgebraic decision procedures have been devised for the related systems KAT and NetKAT [12–14] and have proven quite successful in applications, and we suspect that a similar approach may bear fruit here.

Another interesting direction would be to follow recent work by Joanna Ochremiak [15] involving nominal sets over atoms equipped with both relational and algebraic structure. This is an extension of the original work of Gabbay and Pitts in which atoms can only be compared for equality.

The proof we have provided is concrete and does not explore the rich categorical structure of nominal sets. It would be interesting to rephrase the proof in more abstract terms, which would also be more amenable to generalizations such as those mentioned above.

Acknowledgements

We are grateful to Jamie Gabbay for bringing the original NKA paper to our attention. We would also like to thank Filippo Bonchi, Paul Brunet, Helle Hvid Hansen, Bart Jacobs, Tadeusz Litak, Daniela Petrişan, Damien Pous, Ana Sokolova, and Fabio Zanasi for many stimulating discussions, comments, and suggestions. Finally, we would like to thank the anonymous reviewers for valuable suggestions that significantly improved the presentation and for bringing the work of Laurence and Struth [4] to our attention.

This research was performed at Radboud University Nijmegen and supported by the Dutch Research Foundation (NWO), project numbers 639.021.334 and 612.001.113, the National Security Agency, and the National Science Foundation under grants CCF-1535952 and CCF-1637532.

References

[1] M. Gabbay, A.M. Pitts, A new approach to abstract syntax involving binders, in: 14th IEEE Symp. Logic in Computer Science, 1999, pp. 214–224.
[2] M. Bojanczyk, B. Klin, S. Lasota, Automata theory in nominal sets, Log. Methods Comput. Sci. 10 (3:4) (2014) 1–44, http://dx.doi.org/10.2168/LMCS-10(3:4)2014.
[3] M.J. Gabbay, V. Ciancia, Freshness and name-restriction in sets of traces with names, in: Foundations of Software Science and Computation Structures, 14th International Conference, FOSSACS 2011, in: Lect. Notes Comput. Sci., vol. 6604, Springer, 2011, pp. 365–380.
[4] M.R. Laurence, G. Struth, Completeness theorems for bi-Kleene algebras and series-parallel rational pomset languages, in: P. Höfner, P. Jipsen, W. Kahl, M.E. Müller (Eds.), 14th Int. Conf. Relational and Algebraic Methods in Computer Science, RAMiCS 2014, in: Lect. Notes Comput. Sci., vol. 8428, Springer, 2014, pp. 65–82.
[5] M.J. Gabbay, Foundations of nominal techniques: logic and semantics of variables in abstract syntax, Bull. Symb. Log. 17 (2) (2011) 161–229.
[6] A.M. Pitts, Nominal Sets: Names and Symmetry in Computer Science, Camb. Tracts Theor. Comput. Sci., vol. 57, Cambridge University Press, 2013.
[7] A. Silva, Kleene Coalgebra, Ph.D. thesis, University of Nijmegen, 2010.
[8] E. Kopczynski, S. Torunczyk, LOIS: syntax and semantics, in: Proc. 44th ACM SIGPLAN–SIGACT Symp. Principles of Programming Languages, POPL'17, 2017, pp. 586–598.
[9] B. Klin, M. Szynwelski, SMT solving for functional programming over infinite structures, in: R. Atkey, N. Krishnaswami (Eds.), Proc. 6th Workshop on Mathematically Structured Functional Programming, MFSP 2016, in: EPTCS, vol. 207, Open Publishing Association, 2016, pp. 57–75.
[10] J. Moerman, M. Sammartino, A. Silva, B. Klin, M. Szynwelski, Learning nominal automata, in: Proc. 44th ACM SIGPLAN–SIGACT Symp. Principles of Programming Languages, POPL'17, 2017, pp. 613–625.
[11] D. Kozen, K. Mamouras, D. Petrisan, A. Silva, Nominal Kleene coalgebra, in: M.M. Halldórsson, K. Iwama, N. Kobayashi, B. Speckmann (Eds.), Proc. 42nd Int. Colloq. Automata, Languages, and Programming, Part II, ICALP 2015, in: Lect. Notes Comput. Sci., vol. 9135, EATCS, Springer, Kyoto, Japan, 2015, pp. 290–302.
[12] F. Bonchi, D. Pous, Checking NFA equivalence with bisimulations up to congruence, in: Proc. 40th ACM SIGPLAN–SIGACT Symp. Principles of Programming Languages, POPL'13, ACM, 2013, pp. 457–468.

[13] N. Foster, D. Kozen, M. Milano, A. Silva, L. Thompson, A coalgebraic decision procedure for NetKAT, in: Proc. 42nd ACM SIGPLAN–SIGACT Symp. Principles of Programming Languages, POPL'15, ACM, Mumbai, India, 2015, pp. 343–355.
[14] D. Pous, Symbolic algorithms for language equivalence and Kleene algebra with tests, in: Proc. 42nd ACM SIGPLAN–SIGACT Symp. Principles of Programming Languages, POPL'15, ACM, Mumbai, India, 2015, pp. 357–368.
[15] J. Ochremiak, Nominal sets over algebraic atoms, in: P. Höfner, P. Jipsen, W. Kahl, M.E. Müller (Eds.), 14th Int. Conf. Relational and Algebraic Methods in Computer Science, RAMiCS 2014, in: Lect. Notes Comput. Sci., vol. 8428, Springer, 2014, pp. 429–445.