A formal exploration of Nominal Kleene Algebra

Paul Brunet, Damien Pous

A formal exploration of Nominal Kleene Algebra.

Paul Brunet[1] **and Damien Pous**[*1]

1 Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP

Abstract

An axiomatisation of Nominal Kleene Algebra has been proposed by Gabbay and Ciancia, and then shown to be complete and decidable by Kozen et al. However, one can think of at least four different formulations for a Kleene Algebra with names: using freshness conditions or a presheaf structure (types), and with explicit permutations or not. We formally show that these variations are all equivalent.

Then we introduce an extension of Nominal Kleene Algebra, motivated by relational models of programming languages. The idea is to let letters (i.e., atomic programs) carry a set of names, rather than being reduced to a single name. We formally show that this extension is at least as expressive as the original one, and that it may be presented with or without a presheaf structure, and with or without syntactic permutations. Whether this extension is strictly more expressive remains open.

All our results were formally checked using the Coq proof assistant.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages, F.4.1 Mathematical Logic

Keywords and phrases Nominal sets, Kleene algebra, equational theory, Coq

1 Introduction

Gabbay and Ciancia introduced a nominal extension of Kleene algebra [3], as a framework for trace semantics with dynamic allocation of resources. The associated semantics extends formal languages into nominal languages, where words have a nominal structure. Kozen et al. recently proved the completeness of the proposed axiomatisation [6], and proposed a coalgebraic treatment [5] yielding decidability of the equational theory.

They use the following syntax for nominal regular expressions:

$$e, f ::= a \in \Sigma \mid 0 \mid 1 \mid e + f \mid e \cdot f \mid e^\star \mid \nu a.e \;,$$

where Σ is the alphabet, and $\nu a.e$ makes it possible to generate a fresh letter (or name) a before continuing as e. For instance, the expression $\nu a.\nu b.(a{\cdot}b)$ denotes the language of all words of length two consisting of two distinct letters.

While such a syntax is natural from a nominal point of view, other choices are possible. For instance, one might expect expressions to be typed or classified according to their set of free names. Similarly, name permutations, which are available in any model, can be reified at the syntactic level. We first list four axiomatisations of the corresponding presentations—one of them corresponding to Gabbay and Ciancia's axiomatisation, and we prove that all choices are in fact equivalent. For the sake of the proofs, we need to introduce the *positive* fragments, where the constant 0 denoting the empty language is excluded; these fragments are interesting

[*] This author was funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157); this work has also been supported by the project ANR 12IS02001 PACE.

because they are *stable*: any proof of an equality whose members belong to the fragment only uses expressions from that fragment.

Kleene algebra are known to be complete not only with respect to language models, but also relational models. There, the letters from the alphabet are interpreted as binary relations, and the regular operations correspond to standard operations on binary relations: union, composition, reflexive transitive closure. This makes it possible to interpret imperative programs, seen as state transformers: binary relations between memory states. Kozen actually designed an extension of Kleene algebra, Kleene algebra with tests [4], which makes it possible to represent not only the control flow of such programs, but also the tests and conditions appearing in while loops and branching statements.

Extending Kleene algebra with names seems appropriate to model imperative programs with local variables, where parts of the memory is visible only locally. The previous notion of nominal Kleene algebra is however not really appropriate for this purpose: letters of the alphabet (i.e., atomic programs, instructions) are equated with names bound by the $\nu a.e$ construct (i.e., memory locations). In contrast, the instruction $x \leftarrow y$ that assigns to variable x the value of variable y should be an elementary construction depending on the names x and y. For this reason, we provide an extension of the syntax where letters carry a list of names. (We could use arbitrary nominal sets, but we restrict to a concrete representation in this work for the sake of simplicity.) The typed version of this extension is more appropriate for modelling imperative programs with local variables; like above for plain nominal Kleene algebra, we show that the various presentations are equivalent. We moreover show that the extension is conservative: plain nominal Kleene algebra can be encoded into the extended ones. Whether a converse encoding is possible remains open.

Outline.

We define the various theories in Section 2 and we compare them in Section 3. In Section 4 we provide a relational interpretation for our extended model. We conclude in Section 5.

Notation.

We write $\wp_f(A)$ for the set of finite subsets of A. The set of natural numbers is written $\mathbb{N}$.Composition of two functions f and g is written $f \circ g$; it maps x to $f(g(x))$.

2 Expressions and proofs

2.1 Atoms and letters

Let $\mathcal{A}$ be an infinite set of *atoms* with decidable equality. We consider in this paper *finitely supported permutations of atoms*, simply called permutations in the following. They are bijections π such that there is a finite set $A \subseteq \mathcal{A}$ such that $a \notin A \Rightarrow \pi(a) = a$. The *inverse* of a permutation π is written π^{-1}. The *identity permutation* is denoted by $()$, and the permutation exchanging a and b, and leaving every other atom unchanged, is written $(a\ b)$. Finally, if π is a permutation and A is a finite set of atoms, $\pi(A) := \{\pi(a) \mid a \in A\}$ is *the image of A under π*.

We consider as *letters* an arbitrary nominal set $\mathcal{L}$[2, 7], which we assume to be decidable. Such a set is specified through the data of its set of elements, a function $\natural() : \mathcal{L} \to \wp_f(\mathcal{A})$ mapping every element to its *support*, and an *action* of the group of permutations on $\mathcal{L}$.

These functions must satisfy the following axioms:

$$\forall x \in \mathcal{L}, \forall \pi, (\forall a \in \natural(x), \pi(a) = a) \Rightarrow \pi(x) = x. \tag{1}$$

$$\forall x \in \mathcal{L}, \forall \pi, \natural(\pi(x)) = \pi(\natural(x)). \tag{2}$$

$$\forall x \in \mathcal{L}, \forall \pi, \pi', \pi(\pi'(x)) = \pi \circ \pi'(x). \tag{3}$$

2.2 Expressions and sets of expressions

We define a single type for expressions, containing all possible operators, and we define several fragments of it afterwards. Doing so makes it possible to share several definitions, enabling important code-reuse in our proof scripts.

▸ **Definition 1** (Expressions). The set $\mathbb{E}$ of *expressions* is composed of terms formed over the following syntax, where the letter A is a finite set of atoms, π denotes a permutation, a is an atom and l is a letter:

$$e, f \coloneqq 0 \mid 1 \mid e + f \mid e \cdot f \mid e^\star \mid \nu_a.e \mid l \mid a \mid \langle \pi \rangle e \mid \bot_A \mid id_A \mid w_a.e.$$

Product ($\cdot$), sum ($+$) and Kleene star ($\cdot^\star$) are the regular operations, together with the associated constants 0 and 1, ν_a is name restriction. Variables can be either letters l or atoms a. We include a syntactic construction for explicit permutations $\langle \pi \rangle$. The remaining entries ($\bot_A$, id_A, and w_a) are for the presheaf presentation; we discuss them in Section 2.2.2

2.2.1 Untyped expressions

▸ **Definition 2** (Untyped expression). An expression e is *untyped* if it neither contains the operator w_a nor the constants $\bot_A$ and id_A. The set of untyped expressions is written $\mathbb{U}$.

We define freshness only for untyped expressions:

▸ **Definition 3** (Freshness). An atom a *is fresh for* e if the judgement $a \# e$ can be inferred in the following system.

$$\frac{}{a \# 1} \qquad \frac{}{a \# 0} \qquad \frac{a \notin \natural(l)}{a \# l} \qquad \frac{a \neq b}{a \# b} \qquad \frac{a \# e \quad a \# f}{a \# e + f}$$

$$\frac{a \# e \quad a \# f}{a \# e \cdot f} \qquad \frac{a \# e}{a \# e^\star} \qquad \frac{}{a \# \nu_a.e} \qquad \frac{a \# e}{a \# \nu_b.e} \qquad \frac{\pi^{-1}(a) \# e}{a \# \langle \pi \rangle e}\ .$$

Accordingly, the *support* of an untyped expression e, written $\natural(e)$, is the unique set such that $\forall a,\ a \# e \Leftrightarrow a \notin \natural(e)$.

2.2.2 Typed expressions

For the presheaf approach, we replace freshness assumptions with type derivations. In order to enforce uniqueness of types, we replace the constants 0 and 1 from the untyped syntax by the annotated constants $\bot_A$ and id_A, and we use explicit weakenings (w_a).

▸ **Definition 4** (Typed expressions). For any $e \in E$ and $A \in \wp_f(\mathcal{A})$, e *has the type* A if the judgement $e : A$ can be inferred in the following system:

$$\frac{}{id_A : A} \qquad \frac{}{\bot_A : A} \qquad \frac{l \in \mathcal{L}}{l : \natural(l)} \qquad \frac{a \in \mathcal{A}}{a : \{a\}} \qquad \frac{e : A \quad f : A}{e + f : A}$$

$$\frac{e : A \quad f : A}{e \cdot f : A} \qquad \frac{e : A}{e^\star : A} \qquad \frac{e : A \cup \{a\} \quad a \notin A}{\nu_a.e : A} \qquad \frac{e : A \smallsetminus \{a\} \quad a \in A}{w_a.e : A} \qquad \frac{e : \pi^{-1}(A)}{\langle \pi \rangle e : A}$$

If this is the case, then e is *typed.* The set of typed expressions is written $\mathbb{T}$.

▸ Remark. This type system is syntax directed and yields a simple decision procedure.

2.2.3 Expressions over letters or atoms

A significant motivation for this work was to study the differences between having atoms or letters as variables in expressions. Hence we define two other subsets.

▸ **Definition 5** (Atomic expressions, literate expressions)**.** An expression e is called *atomic* (respectively *literate*) if it does not contain letters (respectively atoms) as variables. The set of atomic expressions is $\mathbb{E}\langle\mathcal{A}\rangle$, and the set of literate expressions is $\mathbb{E}\langle\mathcal{L}\rangle$.

Intuitively, there are two main differences between having atoms or letters as variables. First, a letter may depend on many atoms. Second, two letters with the same support can still be different, whereas the following equivalence holds :

$$\forall a, b \in \mathcal{A},\ a = b \Leftrightarrow (\forall c \in \mathcal{A},\ c \# a \Leftrightarrow c \# b)\,.$$

2.2.4 Positive expressions

For the sake of proofs, we also define the classes of expressions without 0 or $\bot_A$ as a sub-expression. A motivation for excluding these is that in any reasonable system $0 \equiv 0 \cdot e$. Hence if there is an atom a not fresh for e, we would have two equivalent expressions with different sets of fresh variables.

▸ **Definition 6** (Positive expression)**.** An expression e is *positive* if it does not contain 0 nor $\bot_A$ as a sub-expression. The set of positive expressions is $\mathbb{E}^+$.

For concision, we write $\mathbb{E}^+\langle\mathcal{L}\rangle$ for $\mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{E}^+$, and $\mathbb{E}^+\langle\mathcal{A}\rangle$ for $\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{E}^+$.

2.2.5 Explicit permutations

Our syntax includes for explicit permutations $\langle\pi\rangle$, while permutations are usually considered as external operations. This allows one to manipulate permutations inside the expressions, and we shall see that this addition does not raise the complexity of the problem.

Nevertheless, we need to formally define the semantics of permutations on expressions.

▸ **Definition 7** (Action of a permutation on an expression)**.** Let $e \in \mathbb{E}$ be an expression and π a permutation. The *action of* π *on* e, written $\pi \bowtie e$, is defined as follows:

$$\begin{array}{ccc}
\pi \bowtie 1 := 1 & \pi \bowtie 0 := 0 & \pi \bowtie (w_a.e) := w_{\pi(a)}.\,(\pi \bowtie e) \\
\pi \bowtie id_A := id_{\pi(A)} & \pi \bowtie \bot_A := \bot_{\pi(A)} & \pi \bowtie (\nu_a.e) := \nu_{\pi(a)}.\,(\pi \bowtie e) \\
\pi \bowtie a := \pi\,(a) & \pi \bowtie l := \pi\,(l) & \pi \bowtie (\langle\pi'\rangle\, e) := \langle\,()\,\rangle\,(\pi \circ \pi') \bowtie e \\
\pi \bowtie (e^\star) := (\pi \bowtie e)^\star & \pi \bowtie (e \cdot f) := \pi \bowtie e \cdot \pi \bowtie f & \pi \bowtie (e + f) := \pi \bowtie e + \pi \bowtie f
\end{array}$$

Expressions without explicit substitutions are called *clean.*

▸ **Definition 8** (Clean expressions)**.** An expression e is *clean* if it never uses the operator $\langle\pi\rangle$. The set of clean expressions is $\mathbb{C}$.

Applying permutations preserves all classes we have listed so far:

$$\frac{Ax \vdash f = e}{Ax \vdash e = f} \qquad \frac{Ax \vdash e = f \quad Ax \vdash f = g}{Ax \vdash e = f} \qquad \frac{}{Ax \vdash e = f}\ (e,f) \in Ax$$

(a) Equivalence and axiom rules.

$$\frac{}{Ax \vdash 0 = 0} \qquad \frac{}{Ax \vdash 1 = 1} \qquad \frac{}{Ax \vdash id_A = id_A} \qquad \frac{}{Ax \vdash \bot_A = \bot_A}$$

$$\frac{}{Ax \vdash l = l} \qquad \frac{}{Ax \vdash a = a} \qquad \frac{Ax \vdash e = g \quad Ax \vdash f = h}{Ax \vdash e + f = g + h} \qquad \frac{Ax \vdash e = g \quad Ax \vdash f = h}{Ax \vdash e \cdot f = g \cdot h}$$

$$\frac{Ax \vdash e = f}{Ax \vdash e^\star = f^\star} \qquad \frac{Ax \vdash e = f}{Ax \vdash \nu_a.e = \nu_a.f} \qquad \frac{Ax \vdash e = f}{Ax \vdash \varpi_a.e = \varpi_a.f} \qquad \frac{Ax \vdash e = f}{Ax \vdash \langle \pi \rangle\, e = \langle \pi \rangle\, f}$$

(b) Congruence rules.

$$\frac{}{Ax \vdash e + f = f + e} \qquad \frac{}{Ax \vdash e + (f + g) = (e + f) + g} \qquad \frac{}{Ax \vdash e + e = e}$$

$$\frac{}{Ax \vdash e \cdot (f + g) = (e \cdot f) + (e \cdot g)} \qquad \frac{}{Ax \vdash (e + f) \cdot e = (e \cdot g) + (f \cdot g)}$$

$$\frac{}{Ax \vdash e \cdot (f \cdot g) = (e \cdot f) \cdot g} \qquad \frac{Ax \vdash f + e \cdot g \leqslant g}{Ax \vdash e^\star \cdot f \leqslant g} \qquad \frac{Ax \vdash f + g \cdot e \leqslant g}{Ax \vdash f \cdot e^\star \leqslant g}$$

(c) Constant-free Kleene algebra axioms.

Table 1 Modular deduction system

▸ **Lemma 9.** *For any subset of expressions S chosen from $\{\mathbb{T}, \mathbb{U}, \mathbb{E}\langle\mathcal{A}\rangle, \mathbb{E}\langle\mathcal{L}\rangle, \mathbb{E}^+, \mathbb{C}\}$, for any permutation π, and for any expression $e \in \mathbb{E}$, $e \in S \Leftrightarrow \pi \bowtie e \in S$. Furthermore, if e has the type A then $\pi \bowtie e : \pi(A)$, and if a is fresh for e then $\pi(a) \# \pi \bowtie e$.*

(Note the equivalence in the first point, which is why we keep a residual empty permutation when we apply a permutation to an expression of the shape $\langle \pi \rangle\, e$.)

2.3 Proofs

A generic framework for proofs

We now describe a generic framework for defining equational theories over $\mathbb{E}$. Given a relation $Ax \subseteq \mathbb{E} \times \mathbb{E}$, we define the judgement $Ax \vdash e = f$ to hold if it can be inferred in the system displayed in Table 1 (where $Ax \vdash e \leqslant f$ is a shorthand for $Ax \vdash e + f = f$). Notice that we have "hardwired" some laws of Kleene Algebra (KA) in this system, on the basis that they should hold for any reasonable equational system for Nominal Kleene Algebra. However, as we have two sets of constants, we cannot put inside the generic system the Kleene Algebra laws dealing with them. For instance when we consider expressions over the untyped syntax, the fact that $e \cdot 1 = e$ will be stated inside Ax. It is a simple matter to check that whatever Ax, the relation $Ax \vdash \_ = \_$ is an equivalence relation and $Ax \vdash \_ \leqslant \_$ is a preorder.

Sets of axioms

In Tables 2-6, we present a number of possible sets of axioms, which may be combined to axiomatise the different subsets we consider. All the axioms displayed here are implicitly universally quantified.

The first groups of axioms correspond to the axioms of KA for 1, declined in a typed and an untyped fashion. We then do the same for 0 and $\bot_A$, first with the KA axioms, and then for its interactions with $\langle\pi\rangle$, ν_a and w_a, always separating between the typed and untyped cases. These sets of axioms for constants are presented in Tables 2 and 3.

We then introduce sets of axioms to handle permutations. The axiom propagating w_a is set apart, as it only makes sense for typed expressions. This group is displayed in Table 4. Notice that no law speaks about zeros, as it already has been dealt with in (3a).

The sets of axioms in Table 5 are simple distributive laws of the restriction and weakening operators. The next group, displayed in Table 6, constitutes the core of the nominal theory of expressions. The untyped axioms are mostly the classic nominal axioms, taken from [6]. The only new axiom here is (6b), where we use syntactic permutations rather than semantic ones. The typed axioms are for the most part straightforward reformulations of the previous ones. Notice that in the typed case, we do not need to use freshness conditions, but rather typing statements. The last law of the set (6f) reflects the fact that for an untyped expression e, if $a \neq b$ then $a \# e \Leftrightarrow a \# \nu_b.e$.

2.4 Theories

A *theory* is given by a relation Ax, listing the axioms, and a set S from which we take expressions. As expressions may be typed or untyped, atomic or literate, clean or not and positive or not, there are 16 theories, listed in Table 7.

Notice that every subset of expressions mentioned in this table is associated with a single theory. In the following, for concision, we may refer to a theory by simply giving its base set. It is also worth mentioning that for every set S, the theories for $\mathbb{E}\langle\mathcal{L}\rangle \cap S$ and $\mathbb{E}\langle\mathcal{A}\rangle \cap S$ use the same set of axioms.

The theory $\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U} \cap \mathbb{C}$ corresponds precisely to the axiomatisation of NKA given in [6]. In our view, the best theory for defining the interpretation of a program would be $\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \mathbb{C}$, but a relational interpretation is best defined in $\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{T}$.

A difficulty is that if we have a theory (S, Ax), with two expressions $e, f \in S$ such that $Ax \vdash e = f$, it may be the case that the proof uses expressions outside of S. This is generally what happens in systems with 0 (or $\bot_A$): if $e \notin S$ and $0 \in S$, then:

$$\frac{Ax \vdash 0 = 0 \cdot e \qquad Ax \vdash 0 \cdot e = 0}{Ax \vdash 0 = 0}$$

This is a bad property when one wants to prove results by structural induction on proofs. This phenomenon disappears with stable theories:

▶ **Definition 10.** A theory (S, Ax) is *stable* if for any expressions $e, f \in \mathbb{E}$ such that $Ax \vdash e = f$, $e \in S$ if and only if $f \in S$.

All of our positive theories (those included in $\mathbb{E}^+$) are stable.

$$\begin{aligned}
e\cdot 1 &= e\\
1\cdot e &= e\\
1+e\cdot e^\star &\leqslant e^\star\\
1+e^\star\cdot e &\leqslant e^\star
\end{aligned}$$

(a) Untyped identity axioms

$$\begin{aligned}
e\cdot id_A &= e && \text{(if } e : A\text{)}\\
id_A\cdot e &= e && \text{(if } e : A\text{)}\\
id_A+e\cdot e^\star &\leqslant e^\star && \text{(if } e : A\text{)}\\
id_A+e^\star\cdot e &\leqslant e^\star && \text{(if } e : A\text{)}
\end{aligned}$$

(b) Typed identity axioms

Table 2 Identity axioms

$$\begin{aligned}
e+0 &= e\\
e\cdot 0 &= e && \text{(if } e\in\mathbb{U}\text{)}\\
0\cdot e &= e && \text{(if } e\in\mathbb{U}\text{)}\\
\langle\pi\rangle\, 0 &= 0\\
\nu_a.0 &= 0
\end{aligned}$$

(a) Untyped zero axioms

$$\begin{aligned}
e+\bot_A &= e && \text{(if } e : A\text{)}\\
e\cdot\bot_A &= e && \text{(if } e : A\text{)}\\
\bot_A\cdot e &= e && \text{(if } e : A\text{)}\\
\langle\pi\rangle\,\bot_A &= \bot_{\pi(A)}\\
\nu_a.\bot_A &= \bot_{A\smallsetminus\{a\}} && \text{(if } a\in A\text{)}\\
w_a.\bot_A &= \bot_{A\cup\{a\}} && \text{(if } a\notin A\text{)}
\end{aligned}$$

(b) Typed zero axioms

Table 3 Zero axioms

$$\begin{aligned}
\langle\pi\rangle\,(e+f) &= \langle\pi\rangle\, e+\langle\pi\rangle\, f\\
\langle\pi\rangle\,(e\cdot f) &= \langle\pi\rangle\, e\cdot\langle\pi\rangle\, f\\
\langle\pi\rangle\,(e^\star) &= (\langle\pi\rangle\, e)^\star\\
\langle\pi\rangle\,(\nu_a.e) &= \nu_{\pi(a)}.\,\langle\pi\rangle\, e\\
\langle\pi\rangle\,\langle\pi'\rangle\, e &= \langle\pi\circ\pi'\rangle\, e\\
\langle\pi\rangle\, 1 &= 1\\
\langle\pi\rangle\, id_A &= id_{\pi(A)}\\
\langle\,()\,\rangle\, e &= e\\
\langle\pi\rangle\, a &= \pi\,(a) && \text{(if } a\in\mathcal{A}\text{)}\\
\langle\pi\rangle\, l &= \pi\,(l) && \text{(if } l\in\mathcal{L}\text{)}
\end{aligned}$$

(a) General permutation axioms

$$\langle\pi\rangle\,(w_a.e) = w_{\pi(a)}.\,\langle\pi\rangle\, e$$

(b) Permutation vs. w_a

Table 4 Permutation axioms

$$\begin{aligned}
w_a.\,(e+f) &= w_a.e+w_a.f\\
w_a.\,(e^\star) &= (w_a.e^\star)\\
w_a.\,(e\cdot f) &= w_a.e\cdot w_a.f\\
w_a.\,(id_A) &= id_{A\cup\{a\}} && \text{(if } a\notin A\text{)}\\
w_a.\,(w_b.e) &= w_b.\,(w_a.e)
\end{aligned}$$

(a) Weakening

$$\begin{aligned}
\nu_a.\,(e+f) &= \nu_a.e+\nu_a.f\\
\nu_a.\,(\nu_b.e) &= \nu_b.\,(\nu_a.e)
\end{aligned}$$

(b) Restriction

Table 5 Distributive laws of ν_a, w_a

$$\nu_b.e = \nu_a.\,(a\ b) \bowtie e \qquad (\text{if } a \# e)$$

(a) Untyped α-conversion with $\bowtie$

$$\nu_b.e = \nu_a.\langle\,(a\ b)\,\rangle\, e \qquad (\text{if } a \# e)$$

(b) Untyped α-conversion with $\langle\pi\rangle$

$$\nu_b.e = \nu_a.\,(a\ b) \bowtie e \quad (\text{if } \nu_b.e : A \text{ and } a \notin A)$$

(c) Typed α-conversion with $\bowtie$

$$\nu_b.e = \nu_a.\langle\,(a\ b)\,\rangle\, e \quad (\text{if } \nu_b.e : A \text{ and } a \notin A)$$

(d) Typed α-conversion with $\langle\pi\rangle$

$$\begin{aligned} \nu_a.e &= e && (\text{if } a \# e)\\ \nu_a.f \cdot e &= \nu_a.\,(f \cdot e) && (\text{if } a \# e)\\ e \cdot \nu_a.f &= \nu_a.\,(e \cdot f) && (\text{if } a \# e) \end{aligned}$$

(e) Untyped nominal axioms

$$\begin{aligned} \nu_a.w_a.e &= e && (\text{if } \nu_a.w_a.e : A)\\ (\nu_a.f) \cdot e &= \nu_a.\,(f \cdot w_a.e) &&\\ e \cdot (\nu_a.f) &= \nu_a.\,(w_a.e \cdot f) &&\\ \nu_b.w_a.e &= w_a.\nu_b.e && (\text{if } a \neq b) \end{aligned}$$

(f) Typed nominal axioms

Table 6 Nominal axioms

Name	Set	Axioms
NKAmpu	$\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$	(2a) (4a) (5b) (6b) (6e)
NKAspu	$\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$	
NKAnmpu	$\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \mathbb{C}$	(2a) (5b) (6a) (6e)
NKAnspu	$\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U} \cap \mathbb{C}$	
NKAmu	$\mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{U}$	(2a) (3a) (4a) (5b) (6b) (6e)
NKAsu	$\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U}$	
NKAnmu	$\mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \mathbb{C}$	(2a) (3a) (5b) (6a) (6e)
NKAnsu	$\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U} \cap \mathbb{C}$	
NKAmpt	$\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{T}$	(2b) (4a) (4b) (5a) (5b) (6d) (6f)
NKAspt	$\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{T}$	
NKAnmpt	$\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{T} \cap \mathbb{C}$	(2b) (5a) (5b) (6c) (6f)
NKAnspt	$\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{T} \cap \mathbb{C}$	
NKAmt	$\mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{T}$	(2b) (3b) (4a) (4b) (5a) (5b) (6d) (6f)
NKAst	$\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{T}$	
NKAnmt	$\mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{T} \cap \mathbb{C}$	(2b) (3b) (5a) (5b) (6c) (6f)
NKAnst	$\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{T} \cap \mathbb{C}$	

Table 7 Theories

$$\begin{tikzcd}
 & \mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{U} \arrow[rr, dashed, shift right] & & \mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{U} \arrow[ll, shift right] \arrow[rr] \arrow[dl, leftrightarrow] \arrow[dd] & & \mathbb{E}^+\langle\mathcal{L}\rangle\cap\mathbb{U} \arrow[rr, shift left] \arrow[dl, leftrightarrow] \arrow[dd] & & \mathbb{E}\langle\mathcal{L}\rangle\cap\mathbb{U} \arrow[ll, dashed, shift left] \\
\mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{U}\cap\mathbb{C} \arrow[rr, dashed, shift right] & & \mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{U}\cap\mathbb{C} \arrow[ll, shift right] \arrow[dd] & & \mathbb{E}^+\langle\mathcal{L}\rangle\cap\mathbb{U}\cap\mathbb{C} \arrow[rr, shift left] \arrow[dd] & & \mathbb{E}\langle\mathcal{L}\rangle\cap\mathbb{U}\cap\mathbb{C} \arrow[ll, dashed, shift left] & \\
 & \mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{T} \arrow[rr, dashed, shift right] & & \mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{T} \arrow[ll, shift right] \arrow[rr] \arrow[dl, leftrightarrow] \arrow[uu, dashed] & & \mathbb{E}^+\langle\mathcal{L}\rangle\cap\mathbb{T} \arrow[rr, shift left] \arrow[dl, leftrightarrow] \arrow[uu, dashed] & & \mathbb{E}\langle\mathcal{L}\rangle\cap\mathbb{T} \arrow[ll, dashed, shift left] \\
\mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{T}\cap\mathbb{C} \arrow[rr, dashed, shift right] & & \mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{T}\cap\mathbb{C} \arrow[ll, shift right] \arrow[uu, dashed] & & \mathbb{E}^+\langle\mathcal{L}\rangle\cap\mathbb{T}\cap\mathbb{C} \arrow[rr, shift left] \arrow[uu, dashed] & & \mathbb{E}\langle\mathcal{L}\rangle\cap\mathbb{T}\cap\mathbb{C} \arrow[ll, dashed, shift left] &
\end{tikzcd}$$

Figure 1 The two cubes as sets

3 Ordering theories

3.1 Definitions

We define two preorders to compare theories. The first one is the strongest one:

▸ **Definition 11** (Embedding preorder)**.** A theory (S, Ax) *can be embedded into* (S', Ax'), written $(S, Ax) \preccurlyeq (S', Ax')$ if there is a function ϕ such that for any $e \in S$, $\phi(e) \in S'$, and for any $e, f \in S$, $Ax \vdash e = f \Leftrightarrow Ax' \vdash \phi(e) = \phi(f)$. In that case we say that ϕ *is an embedding of* (S, Ax) *into* (S', Ax').

When a theory can be embedded into a second one, then every model of the latter one gives rise to a model former one. However, some intuitively equivalent theory cannot be compared using this preorder. For instance, $\mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{T}$ cannot be embedded into $\mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{U}$. Indeed, while the typed expressions $id_{\{a\}}$ and $id_{\{a,b\}}$ are not equal (they have different types), they have the same untyped behaviour and both of them should be mapped to the untyped constant 1. To this end, we introduce a slightly weaker preorder:

▸ **Definition 12** (Reduction preorder)**.** A theory (S, Ax) *reduces to* (S', Ax'), which we denote by $(S, Ax) \ll (S', Ax')$, if for any pair $(e, f) \in S \times S$ there is a pair of expressions $(e', f') \in S'$ such that $Ax \vdash e = f \Leftrightarrow Ax' \vdash e' = f'$.

▸ **Lemma 13.** *If* $(S, Ax) \ll (S', Ax')$ *and if* (S', Ax') *is decidable, then so is* (S, Ax)*.*

▸ Remark. This lemma assumes an effective proof of $(S, Ax) \ll (S', Ax')$: there must be a way to build the pair e', f' from the pair e, f. Our (Coq) proofs below have this property.

3.2 Embeddings

We summarise the results we obtained using Coq on Figure 1. (The scripts are available online [1]). A plain arrow is drawn between two theories if the source of the arrow can be embedded into the target of the arrow, and a dashed arrow when the source reduces to the target. Thanks to the decidability result for $\mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{U}\cap\mathbb{C}$ [5], this ensures that all atomic theories are decidable.

We discuss in more details how we obtained some of these results.

3.2.1 Reducing to positive fragments

The first step consists in getting rid of the constants 0 and $\bot_A$, so that we can focus on stable theories (Definition 10). We only present here the untyped case. In other words we choose a theory (S, Ax), with S taken from the set $\{\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U}, \mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U} \cap \mathbb{C}, \mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{U}, \mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \mathbb{C}\}$, the corresponding positive theory being $(S \cap \mathbb{E}^+, Ax \setminus (3a))$.

▸ **Definition 14.** If e is an untyped expression, $\mathit{extract}(e)$ is the unique normal form of e with respect to the following confluent rewriting system:

$$e + 0 \to e \qquad 0 + f \to f \qquad e \cdot 0 \to 0 \qquad 0 \cdot f \to 0 \qquad \nu_a.0 \to 0 \qquad \langle\pi\rangle\, 0 \to 0 \qquad 0^\star \to 1.$$

The interesting property of this function is that if $Ax \vdash e = 0$, then $\mathit{extract}(e)$ is syntactically equal to 0, and $\mathit{extract}(e) \in \mathbb{E}^+$ otherwise. Furthermore, for every $e \in S$, $Ax \vdash \mathit{extract}(e) = e$. The formal proof then relies on two key observations:

1. If $(e, f) \in Ax \setminus (3a)$, then $Ax \setminus (3a) \vdash \mathit{extract}(e) = \mathit{extract}(f)$.
2. If $(e, f) \in (3a)$, then $\mathit{extract}(e) = \mathit{extract}(f)$.

This allows to prove by induction on proofs that:

$$Ax \vdash e = f \Rightarrow Ax \setminus (3a) \vdash \mathit{extract}(e) = \mathit{extract}(f)\,.$$

Because the positive axiomatisation is included in Ax, we also get:

$$Ax \setminus (3a) \vdash \mathit{extract}(e) = \mathit{extract}(f) \Rightarrow Ax \vdash \mathit{extract}(e) = \mathit{extract}(f)\,.$$

The fact that $\mathit{extract}(e)$ is provably equal to e with the axioms Ax allows to close the proof of equivalence, with the entailment:

$$Ax \vdash \mathit{extract}(e) = \mathit{extract}(f) \Rightarrow Ax \vdash e = f.$$

However, if $Ax \vdash e = 0$ then $\mathit{extract}(e) \notin \mathbb{E}^+$. This means that we cannot directly use $\mathit{extract}()$ as an embedding between theories. We obtain the reduction as follows: when given the pair e, f, we compute $\mathit{extract}(e)$ and $\mathit{extract}(f)$. If both of these are equal to zero, then when map the pair to equal positive expressions, say $1, 1$. If both of them are non-zero, then we produce $\mathit{extract}(e), \mathit{extract}(f)$. Otherwise we produce different positive expressions, say $1, a$ in the atomic case and $1, l$ in the literate case.

3.2.2 From presheaves to freshness, and back

Let (S, Ax) be a positive untyped theory, meaning $S \subseteq \mathbb{E}^+ \cap \mathbb{U}$, and (S', Ax') be the corresponding positive typed theory. We show here how to transport S into S', and vice versa. This corresponds to the vertical arrows on Figure 1. The key tools in this case are the erasure and retyping functions.

▸ **Definition 15** (Erasure)**.** The *erasure* of $e \in \mathbb{T}$, written $\lfloor e \rfloor$, is the expression obtained from e by removing all weakenings (w_a), and replacing all id_A with 1 and all $\bot_A$ with 0.

▸ **Lemma 16.** *If $e \in S'$ then $\lfloor e \rfloor \in S$, and if $e : A$ then $\natural(\lfloor e \rfloor) \subseteq A$.*

▸ **Definition 17** (Retyping)**.** Let $e \in \mathbb{U}$, we define the retyping of e, written $\lceil e \rceil$, by structural induction:

$$\begin{array}{rcll}
\lceil 0 \rceil := \bot_\varnothing & \lceil 1 \rceil := id_\varnothing & \lceil e + f \rceil := w_{\natural(f)\setminus\natural(e)} \cdot \lceil e \rceil + w_{\natural(e)\setminus\natural(f)} \cdot \lceil f \rceil & \\
\lceil a \rceil := a & \lceil l \rceil := l & \lceil e \cdot f \rceil := w_{\natural(f)\setminus\natural(e)} \cdot \lceil e \rceil \cdot w_{\natural(e)\setminus\natural(f)} \cdot \lceil f \rceil & \\
\lceil e^\star \rceil := \lceil e \rceil^\star & & \lceil \nu_a.e \rceil := \nu_a . \lceil e \rceil & (\text{if } a \in \natural(e)) \\
\lceil \langle\pi\rangle\, e \rceil := \langle\pi\rangle \lceil e \rceil & & \lceil \nu_a.e \rceil := \nu_a . w_a . \lceil e \rceil & (\text{if } a \notin \natural(e))
\end{array}$$

The notation $w_A.e$ is justified by the law $w_a.w_b.e = w_b.w_a.e$, holding in every typed theory.

▸ **Lemma 18.** *$e \in S$ entails $\lceil e \rceil \in S'$. Furthermore, $\lceil e \rceil$ has the type $\natural(e)$.*

These functions allow one to go back and forth between S and S':

▸ **Lemma 19.** *If $e \in S$, then $e = \lfloor \lceil e \rceil \rfloor$. If $e : A$, then:*

$$(6f)\ (5a)\ (4b) \vdash e = w_{A \smallsetminus \natural(\lfloor e \rfloor)} \cdot \lceil \lfloor e \rfloor \rceil .$$

Furthermore, if $S \subseteq \mathbb{C}$, we may remove the axiom $(4b)$.

From this lemma, we obtain that the retyping function is an embedding of S into S'. But it also shows a problem for the other direction. For instance the expressions e and $w_a.e$ have different types, and are thus different, but they will be mapped to the same expression. For this reason, we cannot use the erasure function to embed S' into S.

Nevertheless, we can use it to show that S' is simpler than S. Given a pair of expressions $e, f \in S'$, if e and f have the same type, then we produce the pair $\lfloor e \rfloor, \lfloor f \rfloor$ which is equiprovable. If it is not the case, we purposely produce different expressions, as in the previous section.

3.2.3 From atomic to literate

We assume there is an atom $\alpha \in \mathcal{A}$ and an element $\lambda \in \mathcal{L}$ with $\natural(\lambda) = \{\alpha\}$. We show the transformation from $\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$ to $\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$, which corresponds to the central horizontal top arrow on Figure 1. Let NKApu be the set of axioms $(2a), (4a), (5b), (6b), (6e)$ corresponding to the theory of these sets.

▸ **Definition 20** (From atoms to letters.)**.** Given an expression $e \in \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$, we obtain the expression $\downharpoonleft e \downharpoonright \in \mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$ by replacing every atomic variable a by $\langle\,(a\ \alpha)\,\rangle\,\lambda$. We write $\downharpoonleft \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U} \downharpoonright$ for the set of expressions $f \in \mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$, such that there is an expression $e \in \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$ such that $f = \downharpoonleft e \downharpoonright$.

For any expression e, $e \in \downharpoonleft \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U} \downharpoonright$ if and only if each literate variable in e has the identifier x. It is also worth noting that $\downharpoonleft \_ \downharpoonright$ preserves freshness: $a \# e \Leftrightarrow a \# \downharpoonleft e \downharpoonright$. As for typed and untyped expressions, we define an inverse operation.

▸ **Definition 21** (Going back)**.** The inverse operation is only defined on literate expressions whose variables carry the identifier x, and thus have a singleton support. The expression $\upharpoonleft e \upharpoonright$ is then obtained by replacing every variable by the single atom in its support.

The function $\upharpoonleft \_ \upharpoonright$ is the inverse of $\downharpoonleft \_ \downharpoonright$, $\downharpoonleft \_ \downharpoonright$ preserves NKApu-equality, and $\upharpoonleft \_ \upharpoonright$ preserves NKApu-equality on the image of $\downharpoonleft \_ \downharpoonright$.

▸ **Lemma 22.** $\forall e \in \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$, $\mathrm{NKApu} \vdash e = \upharpoonleft \downharpoonleft e \downharpoonright \upharpoonright$.

▸ **Lemma 23.** $\forall e, f \in \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$, $\mathrm{NKApu} \vdash e = f \Rightarrow \mathrm{NKApu} \vdash \downharpoonleft e \downharpoonright = \downharpoonleft f \downharpoonright$.

▸ **Lemma 24.** $\forall e, f \in \mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \downharpoonleft \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U} \downharpoonright$, $\mathrm{NKApu} \vdash e = f \Rightarrow \mathrm{NKApu} \vdash \upharpoonleft e \upharpoonright = \upharpoonleft f \upharpoonright$.

By putting all together, we obtain that $\downharpoonleft \_ \downharpoonright$ is an embedding of $\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$ into $\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$.

4 Relational interpretation of literate expressions

Our main motivation for developing the typed syntax was to define a relational interpretation of expressions. As explained in the introduction, the classical way of interpreting a program as a relation is to consider memory states as functions, associating values to memory cells. A program is then simply a relation between memory states. Furthermore, in most high level programming languages, one cannot access every part of the memory: a variable should be declared before it is used. There are also constructs allowing one to declare a local variable, which is hidden from the rest of the program. Both of these considerations can be encoded by considering functions with a finite domain: the set of memory locations that are visible in the current scope.

Let us be more precise. Consider that the set $\mathcal{A}$ of atoms corresponds to memory locations, and that locations may contain values from an arbitrary set $\mathcal{V}$. A memory state of domain $A \in \wp_f(\mathcal{A})$ is then a function from A to $\mathcal{V}$, and an expression of type A will be interpreted as a binary relation over memory states of domain A (whence the presheaf structure).

Regular operations are interpreted using the standard operations on binary relations; in particular, id_A is interpreted as the identity relation on $\mathcal{V}^A$. To interpret letters, we need to fix an equivariant map ϕ that assign to a letter x a relation between memory states with domain $\natural(x)$. Several choices are possible for the operations of restriction (ν_a) and weakening (w_a), yielding slightly different theories. Here is a possibility which gives rise to a model of our theory: if R is a relation over $\mathcal{V}^A$, then we define

$$\nu_a.R \coloneqq \{(f{\restriction_A}, g{\restriction_A}) \mid (f,g) \in R\} \quad ; \qquad \text{(if } a \in A\text{)}$$
$$w_a.R \coloneqq \{(f,g) \mid (f{\restriction_A}, f{\restriction_A}) \in R \text{ and } f(a) = g(a)\} \quad . \qquad \text{(if } a \notin A\text{)}$$

(Where $f{\restriction_A}$ is the restriction of a function $f \in \mathcal{V}^B$ for some superset B of A.)

Note that this model is not free: for all relations R, S we have $\nu_a.(R \cdot S) \subseteq (\nu_a.R) \cdot (\nu_a.S)$, which is not an inequation provable from the axioms.

Example.

Consider the program $\mathtt{swap}(x,y)$ that exchanges the contents of the variables x and y. The natural implementation of this program is the following: $\mathtt{let}\ t\ \mathtt{in}\ t \leftarrow x; x \leftarrow y; y \leftarrow t$.

The instruction $\mathtt{x} \leftarrow \mathtt{y}$ may be represented by a nominal element $\mathtt{assign(x,y)}$ with support $\{\mathtt{x},\mathtt{y}\}$, and such that $\pi(\mathtt{assign(x,y)}) = \mathtt{assign}(\pi(\mathtt{x}),\pi(\mathtt{y}))$. Accordingly, the program $\mathtt{swap}$ is represented by the following expression, where the location is hidden using a top-level restriction.

$$\nu_{\mathtt{t}}.(\mathtt{assign(t,x)} \cdot \mathtt{assign(x,y)} \cdot \mathtt{assign(y,t)}) \quad .$$

Alternatively, one can obtain an expression with a single letter using explicit permutations: let a_1 and a_2 be two atoms, and set $\mathbb{a} \coloneqq \mathtt{assign}(a_1, a_2)$. The instruction $\mathtt{x} \leftarrow \mathtt{y}$ may be represented by $\langle(\mathtt{x}\ a_1)\ (\mathtt{y}\ a_2)\rangle\mathbb{a}$, and the program swap by

$$\nu_t.(\langle(\mathtt{t}\ a_1)\ (\mathtt{x}\ a_2)\rangle\mathbb{a}) \cdot (\langle(\mathtt{x}\ a_1)\ (\mathtt{y}\ a_2)\rangle\mathbb{a}) \cdot (\langle(\mathtt{y}\ a_1)\ (\mathtt{t}\ a_2)\rangle\mathbb{a})\ .$$

5 Future work

We leave two questions for future work. First, is it possible to reduce the literate theory of nominal Kleene algebra to that of atomic nominal Kleene algebra? If not, is there a free language theoretic model for which we could obtain decidability?

Second, is there a free relational model for our literate theory?

Acknowledgements.

We would like to thank Daniela Petrisan and Alexandra Silva for the discussions that have led to this work.

References

1 Paul Brunet. Web appendix to this abstract, 2016. `http://perso.ens-lyon.fr/paul.brunet/nka`.

2 Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*, pages 214–224. IEEE, 1999.

3 Murdoch James Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *Foundations of Software Science and Computational Structures, FoSSaCS 2011*, pages 365–380. Springer, 2011.

4 Dexter Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997. `doi:10.1145/256167.256195`.

5 Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan, and Alexandra Silva. Nominal Kleene Coalgebra. In *Automata, Languages, and Programming, ICALP 2015*, pages 286–298. Springer, 2015.

6 Dexter Kozen, Konstantinos Mamouras, and Alexandra Silva. Completeness and incompleteness in Nominal Kleene Algebra. In *Relational and Algebraic Methods in Computer Science, RAMiCS 2015*, pages 51–66. Springer, 2015.

7 Andrew M Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57. Cambridge University Press, 2013.