Unifying Nominal Unification*

Christophe Calvès

Université de Lorraine, LORIA, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France
CNRS, LORIA, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France
Inria, Villers-lès-Nancy, F-54600, France
christophe.calves@gmail.com

Abstract

Nominal unification is proven to be quadratic in time and space. It was so by two different approaches, both inspired by the Paterson-Wegman linear unification algorithm, but dramatically different in the way nominal and first-order constraints are dealt with.

To handle nominal constraints, Levy and Villaret introduced the notion of replacing while Calvès and Fernández use permutations and sets of atoms. To deal with structural constraints, the former use multi-equations in a way similar to the Martelli-Montanari algorithm while the later mimic Paterson-Wegman.

In this paper we abstract over these two approaches and genralize them into the notion of modality, highlighting the general ideas behind nominal unification. We show that replacings and environments are in fact isomorphic. This isomorphism is of prime importance to prove intricate properties on both sides and a step further to the real complexity of nominal unification.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases unification, nominal, α-equivalence, term-graph rewriting

Digital Object Identifier 10.4230/LIPIcs.RTA.2013.143

1 Introduction

Operators with binding are ubiquitous in computer science. Programs, logic formulas, types and proofs are some examples of systems that involve binding. Program transformations, optimisations and compilation, for instance, are defined as operations on programs and should work uniformly on α-equivalence classes. Manipulating terms up to α-equivalence is not easy [8]. Gabbay and Pitts introduced nominal syntax [7, 11] to represent, in a simple and natural way, systems that include binders by extending first-order syntax to provide support for binding operators.

The nominal approach to the representation of systems with binders is characterised by the distinction, at the syntactical level, between atoms (or object-level variables), which can be abstracted (we use the notation $a.t$, where a is an atom and t is a term), and meta-variables (or just variables), which behave like first-order variables but may be decorated with atom permutations. Permutations are generated using swappings (e.g., $(a\ b){\cdot}t$ means swap a and b everywhere in t). For instance, $(a\ b){\cdot}\lambda a.a = \lambda b.b$, and $(a\ b){\cdot}\lambda a.X = \lambda b.(a\ b){\cdot}X$ where λ is here a function symbol (we will introduce the notation formally in the next section). As shown in this example, permutations suspend on variables. The idea is that when a substitution is applied to X in $(a\ b){\cdot}X$, the permutation will be applied to the term that

* A version including every proof is available online at http://christophe.calves.me

24th International Conference on Rewriting Techniques and Applications (RTA'13).
Editor: Femke van Raamsdonk; pp. 143–157
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
RTA

instantiates X. Permutations of atoms are one of the main ingredients in the definition of α-equivalence for nominal terms.

Urban, Pitts and Gabbay [13] showed that *nominal unification*, i.e. unification up to α-equivalence, is decidable. They gave an algorithm to find the most general solution to a nominal unification problem, if one exists. A naive implementation, representing terms as trees, is exponential. Cheney proved that a more general form, called *equivariant unification*, is NP-complete [5]. Fortunately nominal unification was proven to be polynomial [5] and even quadratic [9] using reduction to *higher-order patterns* [12].

Levy and Villaret [10], and Calvès and Fernández [1, 2] presented, independently, two very different algorithms to solve nominal unification in quadratic time and space. Both were inspired by the Paterson-Wegman first-order unification algorithm. But they dramatically differ in the way nominal constraints and equations are dealt with: Levy and Villaret use *replacings* (i.e. sequences of abstractions) and multi-equations rewriting system while Calvès and Fernández use *environments* (i.e. permutations and sets of atoms). While being two very different structures, environments and replacings share the same goal: representing constraints generated by abstractions. The actual complexity of nominal unification is still unknown. Could another representation for nominal or first-order constraints lead to a more efficient algorithm? To answer this question we need to abstract over technical details such as representation of these constraints.

Our constributions are:

- We show that the algorithms in [10] and [1, 2] can be unified. The result is a general abstract nominal-unification algorithm that unifies any algorithm based on Paterson-Wegman first-order linear unification and nominal constraints.
- We develop a general notion of nominal modality to enable reasoning about any data structure representing nominal constraints. We showed that the unification algorithm actually relies on four modality operations, so does its complexity.
- We prove that these strutures are isomorphic. So in particular, environments and replacings are. This also means that any representation can be used to establish properties on any modality. We used this to exchange properties between replacings and environments.

The paper is structured as follows. Section 2 presents nominal terms. Section 3 introduces the notion of *nominal modality* as an abstraction of any data structure used to represent nominal constraints. Section 4 defines operations on modalities. Section 5 establishes properties on any modality. Then section 7 presents the unification algorithm on any modality. Section 8 is about the complexity of modalities' operations. Section 9 discusses related and future work. Finally section 10 concludes.

2 Background

Let Σ be a denumerable set of **function symbols** f, g, $\ldots$; $\mathcal{X}$ a denumerable set of **variables** $X, Y, \ldots$; and $\mathcal{A}$ a denumerable set of **atoms** $a, b, c, d \ldots$ We assume that these sets are pairwise disjoint. In the intended applications, variables will be used to denote meta-level variables (unknowns), and atoms will be used to represent object-level variables, which can be bound (for instance, atoms may be used to represent the variables of a programming language on which we want to reason).

Let Π denote the set of *finite* permutations over $\mathcal{A}$. Its elements are written π,π_i,π',... In this paper, we represent permutations by lists of *swappings* $((a_1\ b_1)\ldots(a_n\ b_n))$. The empty list is written $\mathtt{id}$, inversion and composition are written respectively π^{-1} and $\pi \circ \pi'$. The support is $\mathtt{supp}(\pi) = \{a \in \mathcal{A} \mid \pi(a) \neq a\}$.

$$\frac{}{\gamma \vdash a\#b}\,(\#_{ab}) \qquad \frac{a \# X \in \gamma}{\gamma \vdash a\#X}\,(\#_X)$$

$$\frac{}{\gamma \vdash a\#a.s}\,(\#_{absa}) \qquad \frac{\gamma \vdash a\#s \quad a \neq b}{\gamma \vdash a\#b.s}\,(\#_{absb})$$

$$\frac{\gamma \vdash a\#s_1 \quad \dots \quad \gamma \vdash a\#s_n}{\gamma \vdash a\#f(s_1,\dots,s_n)}\,(\#_f) \qquad \frac{\gamma \vdash \pi^{-1}\cdot a\#X}{\gamma \vdash a\#\pi\cdot X}\,(\#_X)$$

$$\frac{}{\gamma \vdash a \approx a}\,(\approx_a) \qquad \frac{}{\gamma \vdash X \approx X}\,(\approx_X)$$

$$\frac{\gamma \vdash s_1 \approx t_1 \quad \dots \quad \gamma \vdash s_n \approx t_n}{\gamma \vdash f(s_1,\dots,s_n) \approx f(t_1,\dots,t_n)}\,(\approx_f) \qquad \frac{\gamma \vdash ds(\pi,\pi')\#X}{\gamma \vdash \pi\cdot X \approx \pi'\cdot X}\,(\approx_\pi)$$

$$\frac{\gamma \vdash s \approx t}{\gamma \vdash a.s \approx a.t}\,(\approx_{absa}) \qquad \frac{\gamma \vdash s \approx (a\ b)\cdot t \quad a\#t \quad a \neq b}{\gamma \vdash a.s \approx b.t}\,(\approx_{absb})$$

Figure 1 Inductive definition of α-equivalence and freshness relations.

▶ **Definition 1.** The set of *nominal terms*, denoted $\mathcal{N}$, is generated by the grammar

$$s, t, u, v, \dots ::= a \mid \pi\cdot X \mid f(s_1,\dots,s_n) \mid a.s$$

where $a.s$ is an *abstraction* and $\pi\cdot X$ a *suspension*.

For example, $a.a$, $a.f(b, g(a))$ and $a.(X, Y)$ are nominal terms.

The action of a permutation π on a term s, written $\pi\cdot s$, is defined by $\pi\cdot a = \pi(a)$, $\pi\cdot(\pi'\cdot X) = (\pi \circ \pi')\cdot X$, $\pi\cdot a.s = \pi(a).\pi\cdot s$ and $\pi\cdot f(s_1,\dots,s_n) = f(\pi\cdot s_1,\dots,\pi\cdot s_n)$.

▶ **Definition 2 (Subterms).** Let s and t be two nominal terms. $s \preccurlyeq_{\mathcal{N}} t$ means that s is a subterm of t (possibly t itself). $s \prec_{\mathcal{N}} t$ means that s is a strict subterm of t.

Nominal constraints have the form $a\#t$ and $s \approx t$ (read "a fresh for t" and "s α-equivalent to t", respectively). A freshness context γ is a set of freshness constraints of the form $a\#X$. We define the validity of constraints under a freshness context γ inductively, by a system of axioms and rules, using $\#$ in the definition of $\approx$ (see figure 1). In this figure, we write $ds(\pi,\pi')\#X$ as an abbreviation for $\{a\#X \mid a \in ds(\pi,\pi')\}$, where $ds(\pi,\pi') = \{a \mid \pi\cdot a \neq \pi'\cdot a\}$ is the set of atoms where π and π' differ (i.e., their difference set). a, b are any pair of distinct atoms. The relation $\approx$ is indeed an equivalence relation (see [13] for more details).

Let $\mathbb{R} = \mathcal{P}(\mathcal{N}^2)$ be the set of binary relations on $\mathcal{N}$ and $\mathcal{R}_\alpha \in \mathbb{R}$ be the α-equivalence relation on $\mathcal{N}$. Let Δ be the set of *finite* sets of atoms. Let δ,δ_i,δ',... denote elements of Δ.

Substitutions are mappings from variables X to nominal terms, written σ,σ',.... Composition of substitutions is written $\sigma \circ \sigma'$. $t\sigma$ represent the term where every variable X in t has been replaced by $\sigma(X)$.

3 Nominal Modality

We can express the relation $f(s_1,\dots,s_n) \approx f(t_1,\dots,t_n)$ in terms of relations $s_i \approx t_i$. But we can not express $a.s \approx b.t$ in terms of $s \approx t$ where a and b are two different atoms. The relation between s and t is not the α-equivalence but another relation, written $s \approx_{(a\leftarrow b)\cdot e} t$.

This section aims to express any α-equivalence between two nominal terms in terms of (modal) relations between their subterms. The unification algorithm relies on the ability to compute (efficiently) such relations. More precisely, these relations form a family and can be indexed. This section studies those indexes.

3.1 Nominal Pre-Modality

▶ **Definition 3.** Let $\mathcal{R}$ be in $\mathbb{R}$ and a, b be two atoms. Let $(a \leftarrow b)\cdot\mathcal{R} \in \mathbb{R}$ be the relation defined by:

$$(a \leftarrow b)\cdot\mathcal{R} = \{(s,t) \in \mathcal{N}^2 \mid (a.s, b.t) \in \mathcal{R}\}$$

▶ Proposition 1. Let $\mathcal{R}$ be in $\mathbb{R}$ and let $(a_i, b_i)_{i\in\{1\dots n\}}$ be a finite sequence of pair of atoms :

$$(a_n \leftarrow b_n)\cdot \dots (a_1 \leftarrow b_1)\cdot\mathcal{R} = \{(s,t) \in \mathcal{N}^2 \mid (a_1. \dots a_n.s, b_1. \dots b_n.t) \in \mathcal{R}\}$$

▶ **Definition 4.** A *nominal pre-modality*, or *pre-modality* for short, is a set M provided with a function $\Phi_M : M \to \mathbb{R}$ such that:

$$\begin{array}{llll} \exists e \in M, & & \Phi_M(e) = \mathcal{R}_\alpha & (e) \\ \forall m \in M\ \forall a, b \in \mathcal{A}, & \exists m' \in M, & \Phi_M(m') = (a \leftarrow b)\cdot\Phi_M(m) & (\,.\,) \end{array}$$

For example, $\mathbb{R}$ provided with $\Phi_{\mathbb{R}} = \mathtt{id}$ is a pre-modality.

▶ Remark. In the rest of this paper $\Phi_M(m)$ will be written $\mathcal{R}_m$. Furthermore, for any nominal terms s and t, $(s,t) \in \mathcal{R}_m$ will be written $s \approx_m t$.

▶ **Definition 5.** Let M be a pre-modality. Let $\sim$ be the kernel pair of Φ_M, i.e. the equivalence relation on M defined by: $\forall m, m' \in M, \quad m \sim m' \Leftrightarrow \mathcal{R}_m = \mathcal{R}_{m'}$.
The equivalence class of $m \in M$ is written $\overline{m}$.

By abuse of notation, for any pre-modalities M and M', we say that $(m \in M) \sim (m' \in M')$ if $\mathcal{R}_m = \mathcal{R}_{m'}$.

▶ Proposition 2. Let M be a pre-modality, then so is $M/_\sim$ with $\Phi_{(M/_\sim)} = (\Phi_M)/_\sim$. Furthermore, $\Phi_{(M/_\sim)}$ is an injection.

▶ **Corollary 6.** *There is a unique $e \in M/_\sim$ such that $\mathcal{R}_e = \mathcal{R}_\alpha$. e is called the neutral element of $M/_\sim$.*

In the rest of the paper, when it is not ambiguous, e will denote the neutral element of the considered pre-modality.

▶ **Corollary 7.** *Let $m \in M/_\sim$ and $a, b \in \mathcal{A}$. There is a unique $m' \in M/_\sim$ such that $\mathcal{R}_{m'} = (a \leftarrow b)\cdot\mathcal{R}_m$. m' is written $(a \leftarrow b)\cdot m$.*

▶ **Definition 8.** Let M be a pre-modality. $[M]$ is defined as the least subset of $M/_\sim$ such that:

- $e \in [M]$
- $\forall a, b \in \mathcal{A}, \quad m \in [M] \Rightarrow (a \leftarrow b)\cdot m \in [M]$

▶ **Corollary 9.** *Let M be a pre-modality. For any $m \in [M]$ there exists a finite sequence of pair of atoms $(a_i, b_i)_{i\in\{1\dots n\}}$ such that $m = (a_1 \leftarrow b_1)\cdot \dots (a_n \leftarrow b_n)\cdot e$.*

▶ Proposition 3. For any pre-modality M, the restriction $\Phi_{[M]} = (\Phi_{M/_\sim})\mid_{[M]}: [M] \to [\mathbb{R}]$ is a bijection. Furthermore $\Phi_{[M]}(e) = \mathcal{R}_\alpha$ and $\Phi_{[M]}((a \leftarrow b)\cdot m) = (a \leftarrow b)\cdot\Phi_{[M]}(m)$.

3.2 Nominal Modality

▶ **Definition 10.** A *nominal modality* M, or *modality* for short, is a set provided with a bijection $\Phi_M : M \to [\mathbb{R}]$.

- $e = \Phi_M^{-1}(\mathcal{R}_\alpha)$ is called its neutral element.
- For any $a, b \in \mathcal{A}$ and $m \in M$, $(a \leftarrow b)\cdot m = \Phi_M^{-1}((a \leftarrow b)\cdot\mathcal{R}_m)$.

▶ **Corollary 11.** *Let M be a modality, then M is a pre-modality and $[M] \cong M$. Conversely, let M' be a pre-modality, then $[M']$ is a modality.*

▶ Proposition 4. Let M and M' be two modalities. There exist a *unique* bijection $\phi : M \to M'$ such that:

- $\phi(e_M) = e_{M'}$ where e_M (resp. $e_{M'}$) is the neutral element of M (resp. M').
- $\forall a, b \in \mathcal{A}\ \forall m \in M, \quad \phi((a \leftarrow b)\cdot m) = (a \leftarrow b)\cdot\phi(m)$

Such a bijection is called a *nominal-modality isomorphism*, or *modality isomorphism* for short, and is writen $\Phi_{M\to M'}$.

3.3 Environments

This section shows that the set of environments [3] is a pre-modality. Thus, by isomorhism, properties on environments can be transposed to any modality.

▶ **Definition 12.** An *environment* is a pair (π, δ) of a permutation and a set of atoms. The set of environments $(\Pi \times \Delta)$ is written $\mathcal{E}$.

▶ Proposition 5. $\mathcal{E}$ is a pre-modality provided with the function $\Phi_{\mathcal{E}} : \mathcal{E} \to \mathbb{R}$:

$$\mathcal{R}_{(\pi,\delta)} = \{(s,t) \in \mathcal{N}^2 \mid s \approx \pi\cdot t \wedge \delta \# t\}$$

- The neutral element of $\mathcal{E}/_\sim$ is $\overline{(\mathtt{id}, \emptyset)}$.
- for any atoms a and b: $(a \leftarrow b)\cdot\overline{(\pi, \delta)} = \overline{((a\ \pi(b)) \circ \pi, (\delta \cup \{\pi^{-1}(a)\}) \setminus \{b\})}$.

▶ Proposition 6. Let $\xi = (\pi, \delta)$ and $\xi' = (\pi', \delta')$ be two environments:

$$\xi \sim \xi' \Leftrightarrow \begin{cases} \delta = \delta' \\ \mathtt{ds}(\pi, \pi') \subseteq \delta \end{cases}$$

▶ Proposition 7. Let π be a permutation and $\delta = \{a_1, \ldots, a_n\} \in \Delta$ (atoms a_i are considered, without loss of generality, distinct), $a'_i = \pi(a_i)$ and $b \notin \delta$:

$$\forall s, t \in \mathcal{N}, \quad s \approx \pi\cdot t \wedge \delta \# t \Leftrightarrow a'_1.b.\ldots a'_n.b.s \approx \underbrace{b.\ldots b.}_{2n \text{ times}}\pi\cdot t$$

▶ Proposition 8. Let π be a permutation, there exists a finite sequence of pair of atoms $(a_i, b_i)_{i\in\{1\ldots n\}}$ such that: $\forall s, t \in \mathcal{N}, \quad s \approx \pi\cdot t \Leftrightarrow a_1.\ldots a_n.s \approx b_1.\ldots b_n.t$

▶ Proposition 9. For any $\xi = (\pi, \delta) \in \mathcal{E}$ there exists a finite sequence of pair of atoms $(a_i, b_i)_{i\in\{1\ldots n\}}$ such that $\mathcal{R}_{(\pi,\delta)} = (a_1 \leftarrow b_1)\cdot\ldots(a_n \leftarrow b_n)\cdot\mathcal{R}_\alpha$. In other words, $\mathcal{E}/_\sim = [\mathcal{E}]$.

Thus, for any (pre-)modality M and $m \in M$, we can define the freshness set of m using the isomorphism between environments and M. Precisely:

▶ **Definition 13.** For any modality M and any $m \in M$, let $\overline{\xi}(m) \in \mathcal{E}/_\sim$ be the environment equivalence class defined by $\overline{\xi}(m) = \Phi_{M\to\mathcal{E}/_\sim}(m)$. We write $\delta(m)$ for the finite set of atoms and $\Pi(m)$ for the set of permutations such that $\overline{\xi}(m) = \Pi(m) \times \{\delta(m)\}$.

3.4 Simple Replacings

Simple replacings [10], or *replacings* for short, were introduced by Levy and Villaret as a draft for *generalized replacings*. Using them to handle nominal constraints is inefficient in practice but being sequences of abstraction they are useful on the theoritical side.

▶ **Definition 14.** A replacing ℓ is a finite sequence of pair of atoms written $(a_i \leftarrow b_i)_{i\in\{1\dots n\}}$. The set of replacings is written $\mathcal{L}$.

▶ **Definition 15** (Concatenation)**.** Let $\ell = (a_i \leftarrow b_i)_{i\in\{1\dots n\}}$ and $\ell' = (a'_i \leftarrow b'_i)_{i\in\{1\dots n'\}}$ be two replacings. The concatenation of ℓ and ℓ', written $\ell :: \ell'$, is the sequence $(a''_i \leftarrow b''_i)_{i\in\{1\dots(n+n')\}}$ with

$$a''_i \texttt{ (resp. } b''_i\texttt{)} = \begin{cases} a_i \texttt{ (resp. } b_i\texttt{)} & \text{if } 1 \le i \le n \\ a'_{i-n} \texttt{ (resp. } b'_{i-n}\texttt{)} & \text{otherwise} \end{cases}$$

The empty sequence is written ε.

▶ Proposition 10. $\mathcal{L}$ provided with $\Phi_{\mathcal{L}} : \mathcal{L} \to \mathbb{R}$ defined by

$$\Phi_{\mathcal{L}}((a_i \leftarrow b_i)_{i\in\{1\dots n\}}) = \{(s,t) \in \mathcal{N}^2 \mid a_n.\dots a_1.s \approx b_n.\dots b_1.t\}$$

is a pre-modality. The neutral element of $\mathcal{L}/\!\sim$ is $\overline{\varepsilon}$ and, for any $\ell \in \mathcal{L}$, $(a \leftarrow b)\cdot\overline{\ell} = \overline{(a \leftarrow b) :: \ell}$.

▶ Proposition 11. For any $\ell = (a_i \leftarrow b_i)_{i\in\{1\dots n\}} \in \mathcal{L}$, $\overline{\ell} = (a_1 \leftarrow b_1)\cdot\dots(a_n \leftarrow b_n)\cdot\overline{\varepsilon}$. In other words, $\mathcal{L}/\!\sim = [\mathcal{L}]$.

▶ **Definition 16.** For any modality M and any $m \in M$, we write $\overline{\ell}(m) \in \mathcal{L}/\!\sim$ the replacing defined by $\overline{\ell}(m) = \Phi_{M\to\mathcal{L}/\sim}(m)$.

4 Operations

This section presents all the operations on modal relations (and their modality conterparts) used in the unification algorithms.

4.1 Transposition

▶ **Definition 17.** Let $\mathcal{R} \in \mathbb{R}$ be a relation. The transpose of $\mathcal{R}$, written ${}^t\mathcal{R}$, is defined by

$${}^t\mathcal{R} = \{(s,t) \in \mathcal{N}^2 \mid (t,s) \in \mathcal{R}\}$$

▶ **Definition 18.** For any modality M and $m \in M$ we can define the transpose of m, written ${}^tm$, as ${}^tm = \Phi_{[\mathbb{R}]\to M}({}^t\mathcal{R}_m)$.

▶ Proposition 12. Let $\ell = (a_i \leftarrow b_i)_{i\in\{1\dots n\}} \in \mathcal{L}$ be a replacing, ${}^t\ell = (b_i \leftarrow a_i)_{i\in\{1\dots n\}}$.

▶ Proposition 13. For any modalities M and M', $\forall m \in M$, $\quad \Phi_{M\to M'}({}^tm) = {}^t(\Phi_{M\to M'}(m))$.

4.2 Support

▶ **Definition 19.** Let $\xi = (\pi, \delta)$ be an environment. The support of ξ, written, $\mathtt{supp}(\xi)$, is defined as $\mathtt{supp}(\xi) = (\mathtt{id}, \mathtt{supp}(\pi) \cup \delta)$.

▶ Proposition 14. Let $\xi = (\pi, \delta)$ and $\xi' = (\pi', \delta')$ be two environments: $\xi \sim \xi' \Rightarrow \mathtt{supp}(\xi) = \mathtt{supp}(\xi')$. Thus $\mathtt{supp}(\_)$ is well-defined on $\mathcal{E}/\!\sim$: $\mathtt{supp}(\overline{\xi}) = \overline{\mathtt{supp}(\xi)}$.

▶ **Definition 20.** For any modality M and $m \in M$ we can define $\mathtt{supp}(m)$ as

$$\mathtt{supp}(m) = \Phi_{\mathcal{E}/\sim \to M}(\mathtt{supp}(\Phi_{M \to \mathcal{E}/\sim}(m)))$$

▶ Proposition 15. For any modalities M and M'

$$\forall m \in M, \quad \Phi_{M \to M'}(\mathtt{supp}(m)) = \mathtt{supp}(\Phi_{M \to M'}(m))$$

4.3 Monoid

▶ **Definition 21** (Environment composition)**.** Let $\xi = (\pi, \delta)$ and $\xi' = (\pi', \delta')$ be two environments. The composition of ξ and ξ', written $\xi \circ \xi'$, is defined as $\xi \circ \xi' = (\pi \circ \pi', \pi'^{-1}(\delta) \cup \delta')$.

▶ Proposition 16. Let $\xi \in \mathcal{E}$ be an environment and $e = (\mathtt{id}, \emptyset)$ the neutral element of $\mathcal{E}$: $\xi \circ e = e \circ \xi = \xi$.

▶ **Definition 22** (Relation composition)**.** Let $\mathcal{R}, \mathcal{R}' \in \mathbb{R}$ be two relations. We define the composition of $\mathcal{R}$ and $\mathcal{R}'$, written $\mathcal{R} \circ \mathcal{R}'$, as

$$\mathcal{R} \circ \mathcal{R}' = \{(s, u) \in \mathcal{N}^2 \mid \exists t \in \mathcal{N}, \quad (s, t) \in \mathcal{R} \wedge (t, u) \in \mathcal{R}'\}$$

▶ Proposition 17. Let $\xi, \xi' \in \mathcal{E}$ be two environments, then $\mathcal{R}_{\xi \circ \xi'} = \mathcal{R}_{\xi} \circ \mathcal{R}_{\xi'}$.

▶ **Corollary 23.** *Let $\mathcal{R}$ and $\mathcal{R}'$ be two relations in $[\mathbb{R}]$. The following propositions hold:*

- $\mathcal{R} \circ \mathcal{R}_{\alpha} = \mathcal{R}_{\alpha} \circ \mathcal{R} = \mathcal{R}$
- $\mathcal{R} \circ \mathcal{R}' \in [\mathbb{R}]$

So $[\mathbb{R}]$ provided with $\mathcal{R}_{\alpha}$ as its neutral element and the relation composition $(\circ)$ as its internal law is a monoid.

▶ **Definition 24.** Any modality M can be given a monoid structure whose neutral element is e_M and whose internal law $(\circ) : M^2 \to M$ is $\forall m, m' \in M, \quad m \circ m' = \Phi_{[\mathbb{R}] \to M}(\mathcal{R}_m \circ \mathcal{R}_{m'})$.

▶ Proposition 18. For any modalities M and M'

$$\forall m, m' \in M, \quad \Phi_{M \to M'}(m \circ m') = \Phi_{M \to M'}(m) \circ \Phi_{M \to M'}(m')$$

4.4 Replacings operations

▶ **Definition 25** ($\mathtt{For}$)**.** Let M be a modality and $m \in M$, we can define the set of forbidden atoms of m by $\mathtt{For}(m) = \{a \in \mathcal{A} \mid \neg(a \approx_m a)\}$.

▶ **Definition 26** ($\mathtt{Rew}$)**.** Let M be a modality and $m \in M$, we can define the renaming function of m by $\mathtt{Rew}(m) = \{(a \leftarrow b) \in \mathcal{A} \times \mathcal{A} \mid a \neq b \wedge a \approx_m b\}$.

5 Properties

This section presents basic properties on modalities used in the unification algorithm. In the following M denotes a modality and m, m', ... denote elements of M.

5.1 Decomposition

These properties are proven by the isomorphism between M and $\mathcal{E}/\sim$.

▶ Proposition 19. $a.s \approx_m b.t \Leftrightarrow s \approx_{(a \leftarrow b) \cdot m} t$

▶ Proposition 20. $f(s_1, \ldots, s_n) \approx_m f(t_1, \ldots, t_n) \Leftrightarrow s_1 \approx_m t_1 \wedge \cdots \wedge s_n \approx_m t_n$

▶ Proposition 21. $(a\ b) \cdot s \approx_m t \Leftrightarrow s \approx_{((a \leftarrow b) \cdot (b \leftarrow a) \cdot e) \circ m} t$

▶ Proposition 22. $s \approx_m (a\ b) \cdot t \Leftrightarrow s \approx_{m \circ ((a \leftarrow b) \cdot (b \leftarrow a) \cdot e)} t$

5.2 Resolution

Decomposition propositions are able to express relations on nominal terms into relations on their subterms but they are stuck faced to relations such as $X \approx_m X$ or $(s \approx_m t \wedge s \approx_{m'} t)$. This subsection shows how to deal with such cases.

▶ Proposition 23. $s \approx_m s \Leftrightarrow s \approx_{\mathtt{supp}(m)} s$

▶ Proposition 24. $s \approx_m t \wedge t \approx_{m'} u \Leftrightarrow s \approx_m t \wedge s \approx_{m \circ m'} u \Leftrightarrow s \approx_{m \circ m'} u \wedge t \approx_{m'} u$

▶ Proposition 25. $s \approx_m s \wedge s \approx_{m'} t \Leftrightarrow s \approx_{\mathtt{supp}(m) \circ m'} t$

▶ Proposition 26. $s \approx_{{}^t m \circ m} s \Leftrightarrow \delta(m) \# s$

▶ Proposition 27. $m \circ {}^t m \circ m = m$

▶ Proposition 28 (For). $\mathtt{For}(m) = \delta(\mathtt{supp}(m))$

▶ Proposition 29 (Rew). $\mathtt{Rew}(m) = \{(\pi(b) \leftarrow b) \mid b \in \mathtt{supp}(\pi) \setminus \delta(m)\}$ where $\pi \in \Pi(m)$.

6 Modal Problems

▶ **Definition 27** (Equation). A *solution* of the *equation* $s \approx^?_m t$ is a pair (σ, γ) where σ is a substitution and γ freshness context such that $\gamma \vdash s\sigma \approx_m t\sigma$ holds. Two equations are said to be *equivalent* if they have the same set of solutions.

▶ **Definition 28** (Modal Probem). A *nominal modal problem* $\mathcal{P}$ (or *modal problem* for short) is a set of equations of the form $s \approx^?_m t$. (σ, γ) is a *solution* of $\mathcal{P}$ if it is a solution for any equation $s \approx^?_m t$ in $\mathcal{P}$. Two problems are said to be *equivalent* if they have the same set of solutions.

▶ Remark. Note that $s \approx_m t$ is equivalent to $t \approx_{{}^t m} s$. So, in the following, every predicate, pattern, etc involving an equation $s \approx^?_m t$ also matches the equation $t \approx^?_{{}^t m} s$. For example, $a \approx^?_{(c \leftarrow d)\cdot e} b \in \{b \approx^?_{(d \leftarrow c)\cdot e} a\}$.

▶ **Definition 29** (Substitutions). We write $X \mapsto_m s$ for the elementary substitution defined, when $\delta(m) \# X$, by

$$\begin{array}{ll} \sigma(X) = \pi \cdot s & \text{where } \pi \in \Pi(m) \\ \sigma(Y) = Y & \text{otherwise} \end{array}$$

▶ **Definition 30** (Freshness constraints). We write $m \# X$, when $\mathtt{id} \in \Pi(m)$, for the equation $X \approx^?_m X$. Notice that this is equivalent to $\delta(m) \# X$.

The unification algorithm of section 7.2 produces elementary substitution and freshness constraints. They are elementary parts of the incrementatly computed solution. To ease the reading of the paper we have chosen to write them as part of the problem but the reader should keep in mind that they are not inputs but outputs. The "problem" $\{X \mapsto_m s,\ m' \# Y,\ s \approx^?_m t\}$ actually represents the modal problem $\{s \approx^?_m t\}$ where the algorithm has already output the elementary substitution $X \mapsto_m s$ and the freshness constraint $m' \# Y$.

▶ **Definition 31** (Fail). `fail` is to be considered, in the algorithm, as an exception. It means that the problem does not have any solution.

6.1 Graph Representation

▶ **Definition 32.** The *graph representation* of a problem $\mathcal{P}$, written $\mathcal{G}_{\mathcal{P}}$, is the directed graph whose

- vertex (also called *node*) set, written $\mathcal{G}_{\mathcal{P}\mathcal{V}}$ is the set of all nominal terms appearing in the problem: $\mathcal{G}_{\mathcal{P}\mathcal{V}} = \{r \in \mathcal{N} \mid \exists s, t, m \quad r \preccurlyeq_{\mathcal{N}} s \wedge s \approx_m t \in \mathcal{P}\}$.
- edge set, written $\mathcal{G}_{\mathcal{P}\mathcal{E}}$, is $\mathcal{G}_{\mathcal{P}\prec_{\mathcal{N}}} \cup \mathcal{G}_{\mathcal{P}\approx}$ where
  - $\mathcal{G}_{\mathcal{P}\prec_{\mathcal{N}}}$ is the set of *termgraph edges*. Let $s \rightarrowtail_{\mathcal{N}} t$ be such an edge. It means that t is a direct proper subterm of of s, i.e. that either $s = a.t$, $s = f(\ldots, t, \ldots)$ or $s = (a\ b)\cdot t$. s is called a *parent* of t and t is called a *child* of s. In the algorithm, permutation actions $\pi \cdot s$ are not applied but considered as syntactic constructions. For example, let t be the term $(a\ b)(c\ d)\cdot X$. Its child is $(c\ d)\cdot X$ whose child is X.
  - $\mathcal{G}_{\mathcal{P}\approx}$ is the set of equations $s \approx^?_m t$ considered as edges from s to t labelled by m and called *equivalence edge*: $\mathcal{G}_{\mathcal{P}\approx} = \mathcal{P} \cup \{s \approx^?_{(a \leftarrow b)\cdot(b \leftarrow a)\cdot e} t \mid s, t \in \mathcal{G}_{\mathcal{P}\mathcal{V}} \quad s = (a\ b)\cdot t\}$.

▶ Remark. Let $s = (a\ b)\cdot t$ be a term appearing in $\mathcal{P}$. This relation is not only a term edge but also an equivalence relation so it is also represented as an equivalence edge. These relations form the set $\{s \approx^?_{(a \leftarrow b)\cdot(b \leftarrow a)\cdot e} t \mid s, t \in \mathcal{G}_{\mathcal{P}\mathcal{V}} \quad s = (a\ b)\cdot t\}$ which is contained in $\mathcal{G}_{\mathcal{P}\approx}$.

▶ Remark. Several structurally equal subterms can be represented as several nodes but there must be exactly one node per variable in the problem.

▶ Remark. Note that equivalence of equations $s \approx^?_m t$ and $t \approx^?_{{}^t m} s$ implies that an edge from s to t labelled by m is also an edge from t so s labelled by ${}^t m$. Both are considered to be the same edge.

Term edges are good at representing term sharing, but they have the drawback that t is considered as a subterm of $(a\ b)\cdot t$. So a clycle involving of term edge does not implies that the problem have no solution. For example the cycle

$$X \approx^?_e (a\ b) \circ (a\ b)\cdot X,\ (a\ b) \circ (a\ b)\cdot X \rightarrowtail_{\mathcal{N}} (a\ b)\cdot X,\ (a\ b)\cdot X \rightarrowtail_{\mathcal{N}} X$$

is valid. We need a notion of term edges such that an edge from s to t means that t is a subterm of s but s and t can not be equivalent up to a modality:

▶ **Definition 33** (Strict subterm)**.** A *strict term edge* is an edge $s \rightarrowtail_{\not\pi} t$ from s to t such that either $s = a.((a_1\ b_1) \circ \cdots \circ (a_n\ b_n)\cdot t)$ or $s = f(\ldots, (a_1\ b_1) \circ \cdots \circ (a_n\ b_n)\cdot t, \ldots)$. s is called a *strict parent* of t.

If the graph representation of a problem has a cycle involving a strict term edge, then the problem has no solution.

▶ **Definition 34** (Path)**.** Let `Path` be a predicate on pairs of nodes. $\texttt{Path}(s, t) = \top$ if there exists a path from s to t. Let `SPath` be a predicate on pairs of nodes. $\texttt{SPath}(s, t) = \top$ if there exists a *strict path* (a path involving a strict term edge) from s to t.

▶ **Definition 35** (Term Root)**.** Given a problem $\mathcal{P}$, a *term root* is a node in $\mathcal{G}_{\mathcal{P}}$ with no parent. A *strict term root* is a node with no strict parent.

▶ **Definition 36** (Occurences)**.** Given a problem $\mathcal{P}$, the occurence of a node n, written $\texttt{occ}(n)$ is defined as the number of its parents and equivalence arrows involing n:
$\texttt{occ}(n) = |\{t \in \mathcal{G}_{\mathcal{P}\mathcal{V}} \mid t \rightarrowtail_{\mathcal{N}} n \in \mathcal{G}_{\mathcal{P}\prec_{\mathcal{N}}}\}| + |\{(s, m) \in \mathcal{N} \times M \mid s \approx_m t \in \mathcal{G}_{\mathcal{P}\approx}\}|$

▶ **Definition 37.** Given a graph representation $\mathcal{G}_{\mathcal{P}}$. A $\approx$-connected component is a connected component for the graph $(\mathcal{G}_{\mathcal{P}\mathcal{V}}, \mathcal{G}_{\mathcal{P}\approx})$ (same vertices but only equivalence edges).

▶ Remark. $\approx$-connected components represents classes of α-eqvuivalent terms up to a modality.

▶ Remark (Garbage collection). Note that, as a representation of a problem, when a term diseappear from the problem, its node in the graph and all the edges involved disapear too. More precisely, when a term root does not appear in any equivalence edges, this node is garbage collected. All of its children become term roots and may be garbage collected to.

7 Unification Algorithm

Though [10] and [2] both rely on the Paterson-Wegnam first-oder linear algorithm, they use very different approaches. This section shows that even this part of the algorithm can be unified. In addition, thanks to the support operation on modalities, the present algorithm can stay general even when dealing with freshness constraints (see propositions 23 and 26).

7.1 Rules

The following rules never create nominal terms. Instead they rewrite edges of the graph representation of the problem and create elementaty substitutions/freshness constaints.

▶ **Definition 38** (Failure rules).

$$\begin{array}{lcl}
a \approx^?_m f(t_1,\dots,t_n) & \to & \texttt{fail} \\
a \approx^?_m b.t & \to & \texttt{fail} \\
f(s_1,\dots,s_n) \approx^?_m a.t & \to & \texttt{fail} \\
a \approx^?_m b \quad \textbf{when } (a,b) \notin \mathcal{R}_m & \to & \texttt{fail}
\end{array}$$

▶ **Definition 39** (Normalisation rules).

$$\begin{array}{rcll}
s \approx^?_m s & \to & s \approx^?_{\mathsf{supp}(m)} s & \textbf{if } \texttt{id} \notin \Pi(m) \\
s \approx^?_m s, s \approx^?_{m'} t & \to & s \approx^?_{m\circ m'} t & \textbf{if } \texttt{id} \in \Pi(m) \\
s \approx^?_m t, s \approx^?_{m'} t & \to & s \approx^?_{m\circ\mathsf{supp}({}^{t}m'\circ m)} t & \textbf{if } m \neq m'
\end{array}$$

▶ **Definition 40** (Top to Bottom rules).

$$\begin{array}{rcll}
(a\ b)\cdot s \approx^?_m t & \to & s \approx^?_{((a\leftarrow b)\cdot(b\leftarrow a)\cdot e)\circ m} t & \\
\\
a \approx^?_m b & \to & & \textbf{if } (a,b) \in \mathcal{R}_m \\
f(s_1,\dots,s_n) \approx^?_m f(t_1,\dots,t_n) & \to & s_1 \approx^?_m t_1,\dots,s_n \approx^?_m t_n & \\
a.s \approx^?_m b.t & \to & s \approx^?_{(a\leftarrow b)\cdot m} t & \\
\\
X \approx^?_m r & \to & X \mapsto_m r,\ r \approx^?_{{}^{t}m\circ m} r & \textbf{if } \mathtt{occ}(X) = 1 \wedge r \neq X \\
X \approx^?_m X & \to & m \# X & \textbf{if } \mathtt{occ}(X) = 1 \wedge \texttt{id} \in \Pi(m)
\end{array}$$

▶ **Definition 41** ($\approx$-Component Exploration rule). $s \approx^?_m u,\ u \approx^?_{m'} r \to s \approx^?_{m\circ m'} r,\ u \approx^?_{m'} r$

This is clear that all these rules preserve the set of solution as their left-hand sides are equivalent to their right-hand ones as shown in section 5.

7.2 The Paterson-Wegman Strategy

In this section we unify [10] and [2] as a strategy for the rules of section 7.1.

7.2.1 The Strategy

The strategy explores nodes by traversing $\approx$-connected components and assigning to each node n a component representative ($\texttt{repr}(n)$). A component is reduced if all of its nodes are term roots (top to bottom).

▶ **Definition 42.** Let $\mathcal{P}$ be a problem. For any node n, $\texttt{repr}(n)$ represents, if defined ($\perp$ otherwise), an equivalence edge $n \approx^?_m r$ where r is the representative of the $\approx$-connected component of n. Initially $\texttt{repr}(n) = \perp$ for every n. This defines a function $\texttt{repr}$ on nodes treated as a global variable.

We want to reduce first $\approx$-connected components whose nodes are all term roots but looking for one would be inefficient. Instead we process a component until we find a node s which is not a term root. Let p be one of its parents. We need to reduce the component of p first to make s a term root. Any reduction performed on the component of s must wait for the one p to be resolved. We implement this priority system as a representative stack $\mathcal{S}$. Only reduction involving the top element of $\mathcal{S}$ are allowed. This garantees that reductions are performed in the correct order.

▶ **Definition 43.** Let $\mathcal{S}$ be a node stack, treated as a global variable. We define two operations on $\mathcal{S}$: $\texttt{push}(n)$ pushes the node n on the stack and $\texttt{top}$ represents its top element (if it exists). Note that when one node disappear from the problem, it is also removed from $\mathcal{S}$.

This strategy performs stateful computations. The output (written $\mathcal{O}$), the representative function $\texttt{repr}$ and the representative stack $\mathcal{S}$ are global variables so we need to consider a state as a tupple composed of a problem and all of the global variables involved in the strategy. The state generated by Paterson-Wegman Strategy rules verifies some helpful properties so we only consider values of these variables that can be generated by the rules.

▶ **Definition 44.** An ouput, written $\mathcal{O}$, is a set of elementary substitutions and freshness constraints generated by the Paterson-Wegman Strategy rules.

▶ **Definition 45** (State)**.** The set of states $\mathbb{S}$ is defined as the smallet set of 4-tupple $(\mathcal{P}, \mathcal{O}, \texttt{repr}, \mathcal{S})$ (where $\mathcal{P}$ is a problem, $\mathcal{O}$ is an output, $\texttt{repr}$ is a repesentative function and $\mathcal{S}$ is a representative stack) such that :

- for any problem $\mathcal{P}$, the *initial state* is $(\mathcal{P}, \emptyset, (\_ \mapsto \perp), \emptyset) \in \mathbb{S}$.
- if a state $\texttt{st} \in \mathbb{S}$, and $\texttt{st}'$ is obtained by applying one of the Paterson-Wegman Strategy rules on $\texttt{st}$, then $\texttt{st}' \in \mathbb{S}$.

▶ **Definition 46** (Unification Algorithm)**.** Let $\mathcal{P}$ be a problem (the input of the algorithm), take $(\mathcal{P}, \emptyset, (\_ \mapsto \perp), \emptyset)$ as the initial state. Then rewrite it using the following rules until a normal form is reached. If $\texttt{fail}$ is raised, the problem is considered to have no solution. Otherwise consider the output of the normal state as the most-general unifier of the input problem.

Note that the following rules do work on states, but, in order to ease the reading of the paper, patterns use the expressions $s \approx^?_m t$ and $\texttt{repr}(s)$ to represent respectively the equivalence edge from s to t labelled by m and the image of s by the representative function. Similarly, right-hand sides use the expressions $X \mapsto_m r$ and $m \# X$ to represent respectively the addition to the output of the elementary substition and freshness constraint. The expression $\texttt{repr}(s) := s \approx^?_m r$ means that the image of s by the representative function is set to to $s \approx^?_m r$.

Failure rules are applied at every edge creation. This is done in constant time. Normalisation rules are applied mostly on $\texttt{repr}$ at edge creation:

▶ **Definition 47** (Representative Normalisation rules).

$$\begin{array}{rcll}
t \approx^?_m t & \to & t \approx^?_{\mathtt{supp}(m)} t & \textbf{if } \mathtt{id} \notin \Pi(m) \\
s \approx^?_m s,\ s \approx^?_{m'} t & \to & s \approx^?_{m \circ m'} t & \textbf{if } \mathtt{id} \in \Pi(m) \\
s \approx^?_m r,\ \mathtt{repr}(s) & \to & \mathtt{repr}(s) := s \approx^?_{m \circ \mathtt{supp}({}^t m' \circ m)} r & \textbf{if } m \neq m'
\end{array}$$

where $\mathtt{repr}(s) = s \approx^?_{m'} r$. Furthermore, if $s \approx^?_{m'} t$ or $s \approx^?_m s$ is $\mathtt{repr}(s)$ (resp. $t \approx^?_m t$) then $\mathtt{repr}(s)$ becomes $s \approx^?_{m \circ m'} t$ (resp. $t \approx^?_{\mathtt{supp}(m)} t$).

The *top to bottom rules* are only applied on equations $\mathtt{repr}(s)$ when $\mathsf{occ}(s) = 1$:

▶ **Definition 48** (Top to Bottom Representative rules). The following rules have to be applied only when the left-hand side is a $\mathtt{repr}(s)$ for some s such that $\mathtt{occ}(s) = 1$ ($\mathtt{repr}(s) \to \ldots$ **if** $\mathsf{occ}(s) = 1$):

$$\begin{array}{rcll}
(a\ b){\cdot}t \approx^?_m r & \to & t \approx^?_{((a \leftarrow b)\cdot(b \leftarrow a)\cdot e) \circ m} r & \\
 & & & \\
a \approx^?_m b & \to & & \textbf{if } (a, b) \in \mathcal{R}_m \\
f(s_1, \ldots, s_n) \approx^?_m f(r_1, \ldots, r_n) & \to & s_1 \approx^?_m r_1, \ldots, s_n \approx^?_m r_n & \\
a.t \approx^?_m b.r & \to & t \approx^?_{(a \leftarrow b)\cdot m} r & \\
 & & & \\
X \approx^?_m r & \to & X \mapsto_m r,\ r \approx_{{}^t m \circ m} r & \textbf{if } r \neq X \\
X \approx^?_m X & \to & m \# X & \textbf{if } \mathtt{id} \in \Pi(m)
\end{array}$$

where $(a\ b){\cdot}t$, a, $f(s_1, \ldots, s_n)$, $a.t$ and X are instances of s and r, b, $f(r_1, \ldots, r_n)$ and $b.r$ are the representative.

The $\approx$-component exploration rule have to be applied only when r is the representative:

▶ **Definition 49** ($\approx$-Component Representative Definition rule).

$$s \approx^?_m u,\ \mathtt{repr}(u) \ \to \ \begin{cases} (\mathtt{repr}(s) := s \approx^?_{m \circ m'} r),\ \mathtt{repr}(u) & \textbf{if } \mathtt{repr}(s) = \bot \\ \mathtt{fail} & \textbf{if } \mathtt{repr}(s) = s \approx^?_{m''} r' \wedge r' \neq r \\ s \approx^?_{m \circ m'} r,\ \mathtt{repr}(u) & \textbf{if } \mathtt{repr}(s) = s \approx^?_{m''} r \wedge u \neq r \end{cases}$$

where $\mathtt{repr}(u) = u \approx^?_{m'} r$ and $s \notin \{r, u\}$.

The `fail` correspond to the cyclic occurence checking. It occurs when a node s, whose $\mathtt{repr}(s)$ has already been defined as $s \approx^?_{m_1} r_1$ (which means s is in the $\approx$-connected component of r_1), also appear in the $\approx$-connected component of r_2 with $r_1 \neq r_2$. The way representative are selected and defined by traversing term edges make that putting r_1 and r_2 in the same equivalence class would form a cycle.

Finally, we need a rule to initiate the `repr` propagation:

▶ **Definition 50** (`repr` creation).

- If $\mathcal{S} = \emptyset$, then we select a node r, add $\mathtt{repr}(r) = r \approx^?_e r$ to the problem and push r on top of $\mathcal{S}$. r is selected depending on the existance of such a form: if possible, take r of the form $f(\ldots)$, $a.\_$ or an atom, otherwise, take a variable X as r.
- Let s such that $\mathtt{repr}(s) = s \approx^?_m r$, $r = \mathtt{top}$ and s has a strict parent p. Then add $\mathtt{repr}(p) = p \approx^?_e p$ to the problem, fail if $\mathtt{repr}(p) \neq \bot$, and push p on top of $\mathcal{S}$.

7.2.2 Correctness

To prove the correctness, we need to prove that every Paterson-Wegman Strategy is a equivalence on states (so the set of solutions is preserved through rewriting) and that the representative function and stack detect cycles in the solution's graph (occurs-check) (if fail is risen, there is no solution).

▶ Proposition 30. A state is equivalent to the problem

$$\mathcal{P} \cup \{X \approx^?_{(\pi,\emptyset)} s \mid X \mapsto_m s \in \mathcal{O},\ \pi \in \Pi(m)\} \cup \{X \approx^?_m X \mid m \# X \in \mathcal{O}\}$$

By language abuse we call solutions of a state the solutions of its equivalent problem.

▶ Proposition 31. Every Paterson-Wegman Strategy rule transforms a state into an equivalent one or fail.

Proof. Every rule that does not raise fail is an equivalence on the equivalent problem of a state. ◀

Now we need to prove that when fail is raised, then the state does not have any solution.

▶ Proposition 32. Let s be a node such that $\texttt{repr}(s) = s \approx^?_m r$. One step of rewriting either alter the modality m ($\texttt{repr}(s)$ becomes $s \approx^?_{m'} r$) or removes complety s from the problem.

▶ **Corollary 51.** $\texttt{repr}(s) = s \approx^?_m r \Rightarrow \texttt{repr}(r) = r \approx^?_{m'} r \wedge r \in \mathcal{S}$

▶ Proposition 33. $\mathcal{S} = [r_1, \ldots, r_i, \ldots, r_j, \ldots, r_n] \Rightarrow \texttt{SPath}(r_i, r_j)$ where $r_n = \texttt{top}$.

▶ **Corollary 52.** *If $\mathcal{S} = [r_1, \ldots, r_t, \ldots, r_s, \ldots, r_n]$ and $s \approx_m t \in \mathcal{P}$ with $\texttt{repr}(s) = s \approx^?_{m_s} r_s$, $\texttt{repr}(t) = t \approx^?_{m_t} r_t$ and $r_s \neq r_t$. Then the problem has no solution.*

▶ Proposition 34. If fail is raised on a state, then it has no solution.

▶ Proposition 35. An output $\mathcal{O}$ is the most-general unifier of its equivalent problem

$$\{X \approx^?_{(\pi,\emptyset)} s \mid X \mapsto_m s \in \mathcal{O},\ \pi \in \Pi(m)\} \cup \{X \approx^?_m X \mid m \# X \in \mathcal{O}\}$$

7.2.3 Complexity

This section shows that Paterson-Wegman Strategy rules reach a nornal form in a linear number of steps. We consider here one execution of the algorithm on a problem $\mathcal{P}$. The output, repr and $\mathcal{S}$ are treated as global variables.

Formally, let $\mathcal{P}_0$ be a problem. The initial state is $\texttt{st}_0 = (\mathcal{P}_0, \emptyset, (\_ \mapsto \bot), \emptyset)$. Let $[\texttt{st}_0, \texttt{st}_1, \ldots]$ be a sequence of states where $\texttt{st}_{i+1}$ is obtained by applying a Paterson-Wegman Strategy rule on $\texttt{st}_i$.

▶ **Definition 53** (Size). The size of a graph representation $\mathcal{G}_\mathcal{P}$, writen $|\mathcal{G}_\mathcal{P}|$, is defined by

$$|\mathcal{G}_\mathcal{P}| = |\mathcal{G}_{\mathcal{P}\mathcal{V}}| + |\mathcal{G}_{\mathcal{P}\mathcal{E}}|$$

▶ **Definition 54** (Mesure). The mesure of $\mathcal{G}_\mathcal{P}$, written $\mu(\mathcal{G}_\mathcal{P})$, is the sum of

- 1 per node in $\mathcal{G}_{\mathcal{P}\mathcal{V}}$.
- 2 per node whose $\texttt{repr}(s) = \bot$.
- 1 per equivalence edge.
- 1 per equivalence edge $s \approx_m t$ with $\texttt{id} \notin \Pi(m)$.
- 2 per equivalence edge $s \approx_m t$ where $\texttt{repr}(s) \neq s \approx_{m'} t$.
- 3 per term edge.

▶ Proposition 36. $\mu(\mathcal{G}_{\mathcal{P}}) \leq 3|\mathcal{G}_{\mathcal{P}}|$

▶ Proposition 37. The strategy reduces $\mathcal{G}_{\mathcal{P}}$ to a normal form in at most $\mu(\mathcal{G}_{\mathcal{P}})$ steps of rewriting.

▶ **Definition 55.** A state $(\mathcal{P}, \mathcal{O}, \mathtt{repr}, \mathcal{S})$ is said to be in *solved form* either if $\mathcal{P} = \emptyset$ or if `fail` has been raised.

▶ Proposition 38. Normal forms are solved form.

▶ Proposition 39. The algorithm computes the most-general unfier of a problem $\mathcal{P}$ if it exists or raise `fail` in a linear number of rewriting steps.

8 Modal Complexity

The complexity of every rewriting steps depends on the complexity of modal operations which itself depends on the modality used.

Let $\mathcal{P}$ be a problem and $\mathcal{A}_{\mathcal{P}}$ be the set of atoms appearing in $\mathcal{P}$. The unification algorithm does not introduce any atom. So any modal operation computed by any rewriting step only involves atoms in $\mathcal{A}_{\mathcal{P}}$. If any modal operation can be computed in at most $\theta(|\mathcal{A}_{\mathcal{P}}|)$, the complexity of the unification algorithm is at most $\theta(|\mathcal{A}_{\mathcal{P}}| \times \mu(\mathcal{G}_{\mathcal{P}}))$.

As proven in [3], modal operation on environments can be computed in $\theta(|\mathcal{A}_{\mathcal{P}}|)$ using intergers as atoms and arrays as permutations and freshness sets. Thus using environments, the algorithm is quadratic in time.

8.1 Replacings

Computing eagerly replacings would be terribly inefficient. Levy and Villaret avoid this complexity by introducing *generalized replacings* which can be seen as a formulae of modal operations. Using subterm sharing, they get a directed acycling graphs of modal-operation formulas for which they compute the sets $\mathtt{For}(g)$ and $\mathtt{Rew}(g)$ to determine whether $a \approx_g^? b$ is true or not.

▶ **Definition 56** (Generalized Replacings). The set $\mathcal{GL}$ of *generalized replacings* is the set of terms generated by the grammar: $\mathcal{GL} = e \mid (a \leftarrow b){\cdot}\mathcal{GL} \mid {}^t\mathcal{GL} \mid \mathcal{GL} \circ \mathcal{GL} \mid \mathtt{supp}(\mathcal{GL})$.

▶ Remark. The definition in [10] does not contain $\mathtt{supp}(g)$. Instead, multiple occurences of the same variables are kept in multi-equations.

▶ Proposition 40. $\mathcal{GL}$ is a pre-modality.

▶ Proposition 41. $\mathcal{GL}_{/\sim} = [\mathcal{GL}]$

The size of the acyclic graph representing modal-operation formulae is linear in the size of the input problem $\mathcal{P}$ because a linear number of Paterson-Wegman Strategy rules lead to a normal form and each rule involved a bounded number of modal operations. As proven in [10] we can check in quadratic time if the problem has a solution.

9 Related and Future Work

Cheney [4] proved that higher-order pattern unification reduces to nominal unification. Levy and Villaret [9] proved the opposite side. These two results prove that higher-order pattern and nominal unification are equivalent. It would be interesting to adapt modalities to higher-order patterns.

Permissive nominal syntax [6] is a modification to the original syntax to ease the writing of proofs. Given a term t, we know that there are infinitely many fresh atoms for t, but we need freshness constraints to set them. Permissive nominal syntax encodes directly in a term (on variables) which atoms may occur free or not inside. It would be interesting to investigate if our approach can be adapted to take into account this modification.

10 Conclusion

The unique isomorphism between modalities is a powerful tool to establish properties on a representation. Most of the propositions of section 5 where established by proving them on environments and then transposing to any modality by isomorhpism. Furthermore, the algorithm completely isolates nominal constraints from first-order ones. Even when dealing with freshness constraints thanks to Proposition 26.

References

1 Christophe Calvès. *Complexity and Implementation of Nominal Algorithms.* PhD thesis, King's College of London, 2010.

2 Christophe Calvès and Maribel Fernández. The first-order nominal link. In María Alpuente, editor, *LOPSTR*, volume 6564 of *Lecture Notes in Computer Science*, pages 234–248. Springer, 2010.

3 Christophe Calvès and Maribel Fernández. Matching and alpha-equivalence check for nominal terms. *J. Comput. Syst. Sci.*, 76(5):283–301, 2010.

4 J. Cheney. Relating nominal and higher-order pattern unification. In *Proceedings of the 19th International Workshop on Unification (UNIF 2005)*, pages 104–119, 2005.

5 James Cheney. Equivariant unification. *J. Autom. Reasoning*, 45(3):267–300, 2010.

6 Gilles Dowek, Murdoch James Gabbay, and Dominic P. Mulligan. Permissive nominal terms and their unification: an infinite, co-infinite approach to nominal techniques. *Logic Journal of the IGPL*, 18(6):769–822, 2010.

7 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

8 Simon L. Peyton Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, 1987.

9 Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. In Andrei Voronkov, editor, *RTA*, volume 5117 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2008.

10 Jordi Levy and Mateu Villaret. An efficient nominal unification algorithm. In Christopher Lynch, editor, *RTA*, volume 6 of *LIPIcs*, pages 209–226. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

11 Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Inf. Comput.*, 186(2):165–193, 2003.

12 Zhenyu Qian. Linear unification of higher-order patterns. In Marie-Claude Gaudel and Jean-Pierre Jouannaud, editors, *TAPSOFT*, volume 668 of *Lecture Notes in Computer Science*, pages 391–405. Springer, 1993.

13 Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004.