Mathematical Structures in Computer (2019), 1–00
doi:10.1017/xxxxx

CAMBRIDGE
UNIVERSITY PRESS

Formalising Nominal C-Matching Through Unification With Protected Variables

Mauricio Ayala-Rincón[1,3] Maribel Fernández[2] Washington de Carvalho-Segundo[3]
Gabriel Ferreira Silva[3] Daniele Nantes-Sobrinho[1]

Departments of Mathematics[1] and Computer Science[3], University of Brasília (UnB), Brazil
ayala@unb.br wtonribeiro@gmail.com gabrielfsilva1995@gmail.com dnantes@unb.br

[2]Department of Informatics, King's College London, U.K.
maribel.fernandez@kcl.ac.uk

(Received xx xxx xxx; revised xx xxx xxx; accepted xx xxx xxx)

Abstract
This work adapts a formalisation in Coq of a nominal C-unification algorithm specified by a set of non-deterministic rules, to include "protected variables" that cannot be instantiated during the unification process. By introducing protected variables, we are able to reuse the C-unification algorithm to solve nominal C-matching (as well as equality check) problems. From the algorithmic point of view, this extension is sufficient to obtain a C-matching algorithm; however, the resulting algorithm cannot be formally checked by simple reuse of the original formalisation. This paper describes the additional effort necessary in order to adapt the specification of the algorithm and reuse previous formalisations. We also extend a functional recursive nominal C-unification algorithm specified in PVS to include a set of protected variables, effectively adapting this algorithm to the tasks of nominal C-matching and nominal equality check. Experiments comparing executable code extracted from this recursive implementation and a Python manual implementation of the same algorithm are also presented.

Keywords: Nominal Unification; Nominal Matching; Commutative Theory; C-Unification; Formal Methods; Coq; PVS;

1. Introduction

Nominal unification [Urban et al. (2004); Urban (2010)] is the problem of solving equations between nominal terms, that is, terms that include binding operators, in which solutions should be defined modulo α-equivalence. *Nominal matching* is a restriction of nominal unification, in which only one side of the equation can be instantiated. This paper presents a formalisation in Coq of a set of rules for nominal matching and a formalisation in PVS of a recursive nominal matching algorithm, where terms may include commutative operators. This problem is known as nominal C-matching.

In nominal syntax [Pitts (2013)], terms include function symbols, abstractions, and two kinds of variables: *atoms* and *unknowns* (or simply variables). *Atoms* are used to represent object-level variables whereas *unknowns* behave like first-order variables, except that they can have "suspended atom permutations" which act when the variable is instantiated by a term. Atoms can be abstracted over terms, the nominal term $[a]s$ represents the abstraction of a in s and α-equivalence is axiomatised by means of a freshness relation $a\#t$ (read: a is fresh in t) and name-swappings $(a\,b)$, which implement renamings. For example, the first-order logic formula $\forall a.a \geq 0$ can be written as a nominal term $\forall([a]\mathtt{geq}(a,0))$, using function symbols $\forall$ and $\mathtt{geq}$ and an abstracted

atom a. Notice that $\forall([a]\mathtt{geq}(\mathtt{a},\mathtt{X})) \approx^?_\alpha \forall([b]\mathtt{geq}(Y,0))$ is a nominal unification problem whose solution should be such that a does not occur free in Y.

Nominal unification is decidable and efficient nominal unification algorithms are available [Levy and Villaret (2010); Calvès (2010); Calvès and Fernández (2011)], that compute solutions consisting of freshness contexts (containing freshness constraints of the form $a\#X$) and substitutions. Nominal unification modulo commutativity (C) consists of nominal unification problems in which the signature of terms contains commutative function symbols. This algebraic property has to be taken into account during the unification process.

Ayala-Rincón et al. (2018*a*) proposed a nominal C-unification algorithm based on a set of simplification rules. The simplification process generates, for each solvable nominal C-unification problem, a finite set of *fixed point equations* of the form $\pi \cdot X \approx^?_\alpha X$, where π is a permutation and X is a variable, together with a set of *freshness constraints* and a *substitution*. In contrast, the output of the standard nominal unification algorithm consists only of substitutions and freshness constraints. Fixed point equations can be easily eliminated in the standard unification algorithm (they are replaced by freshness constraints), but this is not the case in the presence of commutative symbols. For instance, the fixed point equation $(a\,b)\cdot X \approx^?_{\alpha,C} X$ has infinite solutions $X/a+b, X/(a+b)+(a+b), \ldots$ (see Ayala-Rincón et al. (2017) for a procedure to generate solutions of fixed point equations). A formalisation in Coq of the simplification rules, including correctness and completeness proofs, was later adapted to solve C-matching problems (see Ayala-Rincón et al. (2019)).

In this paper, we revise and extend the previous results, by considering an additional parameter $\mathscr{X}$: a set of *protected variables* that is given as part of the input problem. These variables will not be instantiated during the unification process, therefore the domain of solutions will be disjoint from this set. If the set of protected variables is empty, the algorithm solves general nominal C-unification problems; if the set of protected variables consists of the variables occurring in the right-hand side of equations in the problem given as input, the algorithm outputs a C-matching solution of this problem, if such solution exists; and if the set of protected variables consists of all the variables occurring in the input problem, the algorithm becomes a C-equational checker. Although these conclusions are obvious from the operational point of view, one cannot straightforwardly reuse the formalisation of the nominal C-unification specification to verify the derived nominal C-matching algorithm. In this paper, we show how the formalisation of correctness and completeness of the nominal C-matching specification was reached.

The simplification rules specify an algorithm which is non-deterministic in nature: several rules can apply to the same problem, generating a set of derivations (i.e., a derivation tree), where solutions are at the leaves. To avoid computing the whole derivation tree, in Ayala-Rincón et al. (2019), a recursive functional nominal C-unification algorithm was specified and verified in PVS. In contrast with Ayala-Rincón et al. (2018*a*), where the soundness and completeness of a set of simplification rules was verified, in Ayala-Rincón et al. (2019) the recursive algorithm itself was verified to be correct and complete. This allowed us to obtain executable code from the algorithm. Simpler proofs of termination, soundness and completeness were also obtained, due to the reduction in the number of parameters of the lexicographic measure, from 4 to 2. Additionally, the algorithm was manually translated to obtain a Python 3 implementation.

In this paper we also extend the results in Ayala-Rincón et al. (2019): By adding a parameter for protected variables, the functional algorithm can be used for C-unification (if we pass an empty set as the parameter for protected variables), C-matching (if we pass the variables occurring in the right-hand side of equations in the problem as the parameter for protected variables) or C-equational checking (if we pass all variables in the input problem as the parameter for protected variables). We discuss interesting points of the specification and formalisation of this extension and we also perform experiments comparing the Python manual translation of the algorithm with the code obtained directly from the PVS verified specification.

To summarise, the contributions of this paper are:

- An extension of the nominal C-unification specification proposed by Ayala-Rincón et al. (2018*a*) that adds a parameter for *protected variables*. We show how previous formalisations can be adapted and reused in order to prove termination, soundness and completeness of the simplification rules under this extension.
- An algorithm for nominal C-matching that is obtained by defining the set of protected variables to be the set of variables on the right-hand side of equations in the input problem. Specifications of nominal C-matching problems and solutions are provided and then the set of simplification rules is proven to be terminating, sound and complete.
- An extension of the recursive nominal C-unification algorithm proposed in Ayala-Rincón et al. (2019) that considers protected variables, allowing us to use it for the task of matching and equality check.
- Experiments comparing the code generated by the PVS verified specification and the Python manual implementation of the algorithm.

The proofs presented in this paper were formalised in Coq and in PVS and are available, as well as the Python 3 implementation, at `http://nominal.cic.unb.br/`.

1.1 Related work

Equational unification and matching have been studied in automated reasoning and deduction for more than three decades providing interesting (even open) problems and efficient solutions. For instance, Baader (1986); Baader and Schulz (1996); Baader and Snyder (2001) have investigated several aspects related with general unification with equational theories and combinations of disjoint equational theories, whereas Fages (1987); Kapur and Narendran (1986, 1987, 1992); Siekmann (1979, 1989) have studied associative and/or commutative unification, matching and their complexities.

Nominal equational unification was initially investigated by Ayala-Rincón et al. (2016) and Schmidt-Schauß et al. (2017), using simplification rules to specify unification procedures. In the former paper, the relation of *nominal narrowing* is defined and a *lifting* result relating nominal narrowing and unification is proven, whereas in the latter paper a nominal unification approach for higher order expressions with recursive `let` is presented.

The nominal C-unification algorithm proposed by Ayala-Rincón et al. (2018*a*) outputs a triple consisting of a substitution, a freshness context and a set of fixed point problems, and it was noticed that the set of fixed point equations could generate infinite solutions. In order to give explicit solutions for the fixed point problems, in Ayala-Rincón et al. (2017), combinatorial solutions based on permutation cycles and pseudo-cycles were generated, and an exhaustive search procedure was given. Thus, if solutions are expressed only with freshness contexts and substitutions, nominal C-unification is infinitary, in contrast with standard first-order C-unification which is finitary.

Recently, a new axiomatisation of the alpha equivalence relation for nominal terms was presented [Ayala-Rincón et al. (2018*b*)], which is based on fixed point constraints and allows a finite representation of solutions of nominal C-unification problems, consisting of a substitution and a fixed point context.

In addition to the Coq formalisation of nominal C-unification in Ayala-Rincón et al. (2018*a*), there are formalisations of standard (syntactic) nominal unification in Isabelle [Urban (2010)] and PVS [Ayala-Rincón et al. (2016)]. Regarding formalisations in the (non nominal) standard syntax, Contejean [Contejean (2004)] formalised AC-matching in Coq.

1.2 Organisation

Section 2 presents basic concepts and notations. Section 3 presents the extension of simplification rules for nominal C-unification with protected variables, and discusses how the formalisations of termination, soundness and completeness were adapted. Section 4 introduces the nominal C-matching algorithm and discusses the formalisation of its termination, soundness and completeness properties. Section 5 discusses the functional recursive algorithm for nominal C-unification and the adaptations done to handle a set of protected variables. Section 6 details the experiments comparing the code generated by PVS and the Python implementation. Finally, Section 7 concludes the paper and discusses future work.

2. Background

Consider countable disjoint sets of variables $\mathscr{X} := \{X, Y, Z, \cdots\}$ and atoms $\mathscr{A} := \{a, b, c, \cdots\}$. A *permutation* π is a bijection on $\mathscr{A}$ with a finite *domain*, where the domain (i.e., the *support*) of π is the set $\mathtt{dom}(\pi) := \{a \in \mathscr{A} \mid \pi \cdot a \neq a\}$. We will assume, as in Ayala-Rincón et al. (2019), countable sets of function symbols with different equational properties such as associativity, commutativity, idempotence, etc. Function symbols have superscripts that indicate their equational properties; thus, f_k^C will denote the k^{th} function symbol that is commutative and $f_j^{\emptyset}$ the j^{th} function symbol without any equational property.

Definition 1 (Nominal grammar)**.** *Nominal terms* are generated by the following grammar.

$$s, t := \langle\rangle \mid \bar{a} \mid [a]t \mid \langle s, t\rangle \mid f_k^E\, t \mid \pi.X$$

$\langle\rangle$ denotes the *unit* (that is the empty tuple), $\bar{a}$ denotes an *atom term*, $[a]t$ denotes an *abstraction* of the atom a over the term t, $\langle s, t\rangle$ denotes a *pair*, $f_k^E\, t$ the *application* of f_k^E to t and $\pi.X$ denotes a *moderated variable* or *suspension*. Suspensions of the form $\mathtt{id}.X$ will be represented just by X.

The inverse of π is denoted by π^{-1}. Permutations can be represented by lists of *swappings*, which are pairs of different atoms $(a\ b)$; hence a *permutation* π is a finite list of the form $(a_1\ b_1) :: \ldots :: (a_n\ b_n) :: nil$, where the empty list *nil* corresponds to the identity permutation; concatenation is denoted by $\oplus$ and, when no confusion may arise, $::$ and *nil* are omitted. We follow Gabbay's permutative convention: atoms differ on their names, so for atoms a and b the expression $a \neq b$ is redundant.

Definition 2 (Permutation action)**.** The *action of a permutation* π on a term t, denoted as $\pi \cdot t$, is recursively defined as:

$$\pi \cdot \langle\rangle := \langle\rangle \qquad \pi \cdot \langle u, v\rangle := \langle \pi \cdot u, \pi \cdot v\rangle \qquad \pi \cdot f_k^E\, t := f_k^E\, (\pi \cdot t)$$

$$\pi \cdot \bar{a} := \overline{\pi \cdot a} \qquad \pi \cdot ([a]t) := [\pi \cdot a](\pi \cdot t) \qquad \pi \cdot (\pi'\,.\,X) := (\pi' \oplus \pi)\,.\,X$$

Remark 3. Notice that according to the definition of the action of a permutation over atoms, the composition of permutations π and π', usually denoted as $\pi \circ \pi'$, corresponds to the append $\pi' \oplus \pi$. Also notice that $\pi' \oplus \pi \cdot t = \pi \cdot (\pi' \cdot t)$.

Definition 4 (Difference set)**.** The *difference set* between two permutations π and π' is the set of atoms where the action of π and π' differs: $ds(\pi, \pi') := \{a \in \mathscr{A} \mid \pi \cdot a \neq \pi' \cdot a\}$.

The set of variables occurring in a term t will be denoted as $\mathtt{var}(t)$. This notation extends to a set S of terms in the natural way: $\mathtt{var}(S) = \bigcup_{t \in S} \mathtt{var}(t)$.

A *substitution* σ is a mapping from variables to terms such that $X \neq X\sigma$ only for a finite set of variables. This set is called the *domain* of σ and is denoted by $\mathtt{dom}(\sigma)$. For $X \in \mathtt{dom}(\sigma)$, $X\sigma$ is called the *image* of X by σ. Define the image of σ as $\mathtt{im}(\sigma) = \{X\sigma \mid X \in \mathtt{dom}(\sigma)\}$. The set of variables occurring in the image of σ is then $\mathtt{var}(\mathtt{im}(\sigma))$. A substitution σ with $\mathtt{dom}(\sigma) := \{X_0, \cdots, X_n\}$ can be represented as a set of *binds* in the form $\{X_0/t_0, \cdots, X_n/t_n\}$, where for $0 \leq i \leq n$, $X_i\sigma = t_i$.

Definition 5 (Substitution action)**.** The *action of a substitution* σ on a term t, denoted $t\sigma$, is defined recursively as follows:

$$\langle\rangle\sigma := \langle\rangle \qquad \overline{a}\sigma := \overline{a} \qquad (f_k^E\, t)\sigma := f_k^E\, t\sigma$$

$$\langle s, t\rangle\sigma := \langle s\sigma, t\sigma\rangle \qquad ([a]t)\sigma := [a]t\sigma \qquad (\pi.X)\sigma := \pi \cdot X\sigma$$

Example 6. For term $t = \langle (a\ b).X, f(e)\rangle$ and substitution $\sigma = \{X/[a]a\}$, we obtain that $t\sigma = \langle [b]b, f(e)\rangle \approx_\alpha \langle [a]a, f(e)\rangle$, for some unary function symbol f in the signature.

The following result can be proved by induction on the structure of terms.

Lemma 7 (Substitutions and permutations commute)**.** $(\pi \cdot t)\sigma = \pi \cdot (t\sigma)$

The inference rules defining freshness and α-equivalence are given in Figures 1 and 2. The symbols ∇ and Δ are used to denote *freshness contexts* that are sets of constraints of the form $a\#X$, meaning that the atom a is fresh in X. The domain of a freshness context $dom(\Delta)$ is the set of atoms appearing in it; $\Delta|_X$ denotes the restriction of Δ to the freshness constraints on X: $\{a\#X \mid a\#X \in \Delta\}$. The rules in Figure 1 are used to check if an atom a is fresh in a nominal term t under a freshness context ∇, also denoted as $\nabla \vdash a\#t$.

The rules in Figure 2 are used to check if two nominal terms s and t are α-equivalent under some freshness context ∇, written as $\nabla \vdash s \approx_\alpha t$. These rules use the inference system for freshness constraints: specifically freshness constraints are used in rule $(\approx_\alpha [\mathbf{ab}])$.

Remark 8. $\mathtt{dom}(\pi)\#X$ and $ds(\pi, \pi')\#X$ denote, respectively, the sets $\{a\#X \mid a \in \mathtt{dom}(\pi)\}$ and $\{a\#X \mid a \in ds(\pi, \pi')\}$. Notice that $\mathtt{dom}(\pi) = ds(\pi, id)$.

$$\frac{}{\nabla \vdash a\,\#\,\langle\rangle}\ (\#\langle\rangle) \qquad \frac{}{\nabla \vdash a\,\#\,\overline{b}}\ (\#\,\mathbf{atom}) \qquad \frac{\nabla \vdash a\,\#\,t}{\nabla \vdash a\,\#\,f_k^E\ t}\ (\#\,\mathbf{app}) \qquad \frac{}{\nabla \vdash a\,\#\,[a]t}\ (\#\,\mathbf{a}[\mathbf{a}])$$

$$\frac{\nabla \vdash a\,\#\,t}{\nabla \vdash a\,\#\,[b]t}\ (\#\,\mathbf{a}[\mathbf{b}]) \qquad \frac{(\pi^{-1} \cdot a\#X) \in \nabla}{\nabla \vdash a\,\#\,\pi.X}\ (\#\,\mathbf{var}) \qquad \frac{\nabla \vdash a\,\#\,s \quad \nabla \vdash a\,\#\,t}{\nabla \vdash a\,\#\,\langle s, t\rangle}\ (\#\,\mathbf{pair})$$

Figure 1. Rules for the freshness relation.

Key properties of the nominal freshness and α-equivalence relations have been extensively explored in previous works [Ayala-Rincón et al. (2019); Ayala-Rincón et al. (2016); Urban (2010); Urban et al. (2004)]. Among them we have *freshness preservation*: if $\nabla \vdash a\,\#\,s$ and $\nabla \vdash s \approx_\alpha t$,

$$\frac{}{\nabla \vdash \langle\rangle \approx_\alpha \langle\rangle}\ (\approx_\alpha \langle\rangle) \qquad \frac{}{\nabla \vdash \overline{a} \approx_\alpha \overline{a}}\ (\approx_\alpha \textbf{atom}) \qquad \frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash f_k^E\ s \approx_\alpha f_k^E\ t}\ (\approx_\alpha \textbf{app})$$

$$\frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash [a]s \approx_\alpha [a]t}\ (\approx_\alpha [\textbf{aa}]) \qquad \frac{\nabla \vdash s \approx_\alpha (a\ b) \cdot t \quad \nabla \vdash a \# t}{\nabla \vdash [a]s \approx_\alpha [b]t}\ (\approx_\alpha [\textbf{ab}])$$

$$\frac{ds(\pi, \pi')\#X \subseteq \nabla}{\nabla \vdash \pi.X \approx_\alpha \pi'.X}\ (\approx_\alpha \textbf{var}) \qquad \frac{\nabla \vdash s_0 \approx_\alpha t_0 \quad \nabla \vdash s_1 \approx_\alpha t_1}{\nabla \vdash \langle s_0, t_0\rangle \approx_\alpha \langle s_1, t_1\rangle}\ (\approx_\alpha \textbf{pair})$$

Figure 2. Rules for the relation α-equivalence relation.

then $\nabla \vdash a \# t$; *equivariance*: for all permutations π, if $\nabla \vdash s \approx_\alpha t$ then $\nabla \vdash \pi \cdot s \approx_\alpha \pi \cdot t$; and *equivalence*: $\nabla \vdash \_ \approx_\alpha \_$ is an equivalence relation.

3. A nominal C-unification algorithm with protected variables

In Ayala-Rincón et al. (2018*a*) we proposed a nominal C-unification algorithm which used a set of transformation rules to deal with equations (Figure 4) and another set of rules to deal with freshness constraints and contexts (Figure 3). These rules act over triples of the form $\langle \nabla, \sigma, P\rangle$, where σ is a substitution. In this work, we will deal with nominal C-equational problems including another parameter that is a set $\mathscr{X}$ of protected variables. This parameter indicates which variables are forbidden to be instantiated. The quadruple that will be associated with a C-equational problem of the form $\langle \nabla, \mathscr{X}, P\rangle$ is $\langle \nabla, \mathscr{X}, id, P\rangle$. Calligraphic uppercase letters (e.g., $\mathscr{P}, \mathscr{Q}, \mathscr{R}$, etc) will denote quadruples.

Definition 9 (Unification and matching problem)**.** An *unification problem* is a triple $\langle \nabla, \mathscr{X}, P\rangle$, where ∇ is a *freshness context*, $\mathscr{X}$ a set of protected variables, and P is a finite set of *equations* and *freshness constraints* of the form $s \approx_? t$ and $a\#_? s$, respectively, s and t are terms and a is an atom. Nominal terms in the equations preserve the syntactic restriction that commutative symbols are only applied to pairs. A *matching problem* consists only of a pair $\langle \nabla, P\rangle$.

Definition 10 (Solution for a C-unification problem with preserved variables)**.** A *solution* for a quadruple $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P\rangle$ is a pair $\langle \nabla, \sigma\rangle$, where the domain of σ has no variables in $\mathscr{X}$, and the following conditions are satisfied:

(1) $\nabla \vdash \Delta\sigma$;
(2) if $a\#_? t \in P$ then $\nabla \vdash a \# t\sigma$;
(3) if $s \approx_? t \in P$ then $\nabla \vdash s\sigma \approx_{\{\alpha, C\}} t\sigma$;
(4) there exists λ such that $\nabla \vdash \delta\lambda \approx \sigma$.

A *solution for a C-equational problem with preserved variables* $\langle \Delta, \mathscr{X}, P\rangle$ is a solution for the associated quadruple $\langle \Delta, \mathscr{X}, id, P\rangle$. The *solution set* for a problem or quadruple $\mathscr{P}$ is denoted by $\mathscr{U}_C(\mathscr{P})$.

We will denote the set of variables occurring in the set P of a problem $\mathscr{P}$ or quadruple $\mathscr{P} = \langle \nabla, \mathscr{X}, \sigma, P\rangle$ as $\mathtt{var}(P)$ or $\mathtt{var}(\mathscr{P})$, respectively.

When $\mathscr{X}$ equals $\emptyset$, the notions of C-unification problem with preserved variables, and its solutions coincide with the corresponding notions for C-unification as given in Ayala-Rincón et al. (2018*a*). For simplicity, C-unification problems and solutions with preserved variables will be called just C-unification problems and solutions.

$$(\#_?\langle\rangle)\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{a\#_?\langle\rangle\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P\rangle} \qquad (\#_?\mathbf{a}\bar{\mathbf{b}})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{a\#_?\bar{b}\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P\rangle} \qquad (\#_?\mathbf{app})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{a\#_? f\,t\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{a\#_? t\}\rangle}$$

$$(\langle\#\rangle\mathbf{pair})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{a\#_?\langle s,t\rangle\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{a\#_? s, a\#_? t\}\rangle} \qquad (\#_?\mathbf{a}[\mathbf{a}])\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{a\#_?[a]t\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P\rangle}$$

$$(\#_?\mathbf{a}[\mathbf{b}])\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{a\#_?[b]t\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{a\#_? t\}\rangle} \qquad (\#_?\mathbf{var})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{a\#_?\pi.X\}\rangle}{\langle \{(\pi^{-1}\cdot a)\#X\} \cup \nabla, \mathscr{X}, \sigma, P\rangle}$$

Figure 3. Reduction rules for freshness problems.

$$(\approx_?\mathbf{refl})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{s \approx_? s\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P\rangle} \qquad (\approx_?\mathbf{pair})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{\langle s_1,t_1\rangle \approx_? \langle s_2,t_2\rangle\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{s_1 \approx_? s_2, t_1 \approx_? t_2\}\rangle}$$

$$(\approx_?\mathbf{inv})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{\pi.X \approx_? \pi'.X\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{\pi \oplus (\pi')^{-1}.X \approx_? X\}\rangle},\ \text{if } \pi' \neq \mathtt{id} \qquad (\approx_?\mathbf{app})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{f_k^E\, s \approx_? f_k^E\, t\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{s \approx_? t\}\rangle},\ \text{if } E \neq C$$

$$(\approx_?\mathbf{C})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{f_k^C\, s \approx_? f_k^C\, t\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{s \approx_? v\}\rangle},\ \left\{\begin{array}{l}\text{where } s = \langle s_0, s_1\rangle \text{ and } t = \langle t_0, t_1\rangle\\ v = \langle t_i, t_{i+1}\rangle, i = 0, 1\end{array}\right\}$$

$$(\approx_?[\mathbf{aa}])\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{[a]s \approx_? [a]t\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{s \approx_? t\}\rangle} \qquad (\approx_?[\mathbf{ab}])\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{[a]s \approx_? [b]t\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{s \approx_? (a\,b)\,t, a\#_? t\}\rangle}$$

$$(\approx_?\mathbf{inst})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{\pi.X \approx_? t\} \text{ or } \{t \approx_? \pi.X\}\rangle \ \text{ let } \sigma' := \sigma\{X/\pi^{-1}\cdot t\}}{\left\langle \nabla, \mathscr{X}, \sigma', P\{X/\pi^{-1}\cdot t\} \ \cup \bigcup_{\substack{Y \in dom(\sigma'),\\ a\#Y \in \nabla}} \{a\#_? Y\sigma'\} \right\rangle},\ \text{if } X \notin \mathtt{var}(t) \cup \mathscr{X}$$

Figure 4. Reduction rules for equational problems.

These inference rules transform a nominal unification problem into a finite set of unification problems consisting only of *fixed point* equations, i.e., equations of the form $\pi.X \approx_? X$, together with a substitution and a freshness context. P is called a *fixed point problem* if it is a set of fixed point equations. We will denote by $P_{\approx}$, $P_{\#}$ and $P_{\mathrm{fp}_{\approx}}$ the sets of equations, freshness problems and fixed point equations in the set P of a unification problem $\langle \nabla, \mathscr{X}, P\rangle$.

Also, we use rules described in Figure 5, called *unification steps*, which give a strategy for application of rules specified as presented in Figures 4 and 3.

Differently from Ayala-Rincón et al. (2018*a*), rule $(\approx_?\ \textbf{inst})$, checks whether the variable X is a protected variable, before applying the instantiation.

The set $\mathscr{X}$ has no effect on the rules for freshness and is not altered by any rule. Rules in Figure 4 will be applied without restrictions by use of rule $(\upsilon_\approx)$, but freshness constraints are reduced only when all equations were reduced and the problem consists of fixed point equations. This fact is expressed by the condition $P_\approx = P_{fp_\approx}$ in rule $(\upsilon_\#)$.

$$(\upsilon_\approx)\frac{\mathscr{P} \Rightarrow_\approx \mathscr{Q}}{\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}} \qquad (\upsilon_\#)\frac{\mathscr{P} \Rightarrow_\# \mathscr{Q}}{\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}}\ , P_\approx = P_{fp_\approx}$$

Figure 5. Unification step.

Derivation with rules of Figure 4 is denoted by $\Rightarrow_\approx$; thus, $\langle \nabla, \mathscr{X}, \sigma, P\rangle \Rightarrow_\approx \langle \nabla, \mathscr{X}, \sigma', P'\rangle$ means that the second quadruple is obtained from the first one by application of one rule. We will use the standard rewriting nomenclature, e.g., we will say that $\mathscr{P}$ is a *normal form* or *irreducible* by $\Rightarrow_\approx$, denoted by $\Rightarrow_\approx$*-nf*, whenever there is no $\mathscr{Q}$ such that $\mathscr{P} \Rightarrow_\approx \mathscr{Q}$; $\Rightarrow_\approx^*$ and $\Rightarrow_\approx^+$ denote respectively derivations in zero or more and one or more applications of the rules in Figure 4. Derivation with rules of Figure 3 is denoted by $\Rightarrow_\#$.

The theorem below summarises the termination and correctness results regarding the algorithm proposed.

Theorem 11 (Properties of $\Rightarrow_\approx$, $\Rightarrow_\#$ and $\Rightarrow_\upsilon$)**.**

(1) (Decidability of $\Rightarrow_\approx$, $\Rightarrow_\#$ and $\Rightarrow_\upsilon$) Given a quadruple $\mathscr{P}$, it is possible to decide whether $\mathscr{P}$ is a normal form w.r.t. $\Rightarrow_\approx$ (resp. $\Rightarrow_\#$) or there exists $\mathscr{Q}$ such that $\mathscr{P} \Rightarrow_\approx \mathscr{Q}$ (resp. $\mathscr{P} \Rightarrow_\# \mathscr{Q}$).
(2) (Termination of $\Rightarrow_\approx$, $\Rightarrow_\#$ and $\Rightarrow_\upsilon$) The relations $\Rightarrow_\approx$, $\Rightarrow_\#$ and $\Rightarrow_\upsilon$ are terminating.
(3) (Completeness of $\Rightarrow_\#$) If $\mathscr{P} \Rightarrow_\# \mathscr{Q}$, then $\langle \nabla, \sigma\rangle \in \mathscr{U}_C(\mathscr{P})$ if and only if $\langle \nabla, \sigma\rangle \in \mathscr{U}_C(\mathscr{Q})$.

Proof. The proofs are obtained by adjusting proofs in Ayala-Rincón et al. (2018*a*) taking into account the set of variables $\mathscr{X}$. □

Remark 12. We will call $\mathscr{Q}$ a *leaf* if it is a normal form w.r.t. $\Rightarrow_\upsilon$.

For completeness, we will restrict ourselves to idempotent solutions, in order to obtain such property, we will restrict the applications of rules to *valid* quadruples.

Definition 13 (Valid quadruple)**.** $\mathscr{P} = \langle \nabla, \mathscr{X}, \sigma, P\rangle$ *is valid* if $\mathtt{im}(\sigma) \cap \mathtt{dom}(\sigma) = \emptyset$ and $\mathtt{dom}(\sigma) \cap \mathtt{var}(P) = \emptyset$.

As for Theorem 11 , the formalisations of Lemmas 14, 15, 16, and Theorems 18 and 20 are quite similar to the proofs of corresponding lemmas in Ayala-Rincón et al. (2018*a*). For this reason, they are omitted.

Lemma 14 (Preservation of valid quadruples)**.**

(1) If $\mathscr{P} \Rightarrow_\# \mathscr{Q}$ or $\mathscr{P} \Rightarrow_\approx \mathscr{Q}$, and $\mathscr{P}$ is valid then $\mathscr{Q}$ is also valid.
(2) If $\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}$ and $\mathscr{P}$ is valid then $\mathscr{Q}$ is also valid.

Lemma 15 (Preservation of solutions)**.** Let $\mathscr{P}$ be a valid quadruple.

(1) If $\mathscr{P} \Rightarrow_{\approx} \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$, then $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$.
(2) If $\mathscr{P} \Rightarrow_{\upsilon} \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$, then $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$.

Lemma 16 (Completeness of $\Rightarrow_{\approx}$ and $\Rightarrow_{\upsilon}$)**.** Let $\mathscr{P}$ be a valid quadruple.

(1) If $\mathscr{P}$ is not a normal form w.r.t. $\Rightarrow_{\approx}$, then $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$ if and only if there exists $\mathscr{Q}$ such that $\mathscr{P} \Rightarrow_{\approx} \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$.
(2) If $\mathscr{P}$ is not a leaf, then $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$ if and only if there exists $\mathscr{Q}$ such that $\mathscr{P} \Rightarrow_{\upsilon} \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$.

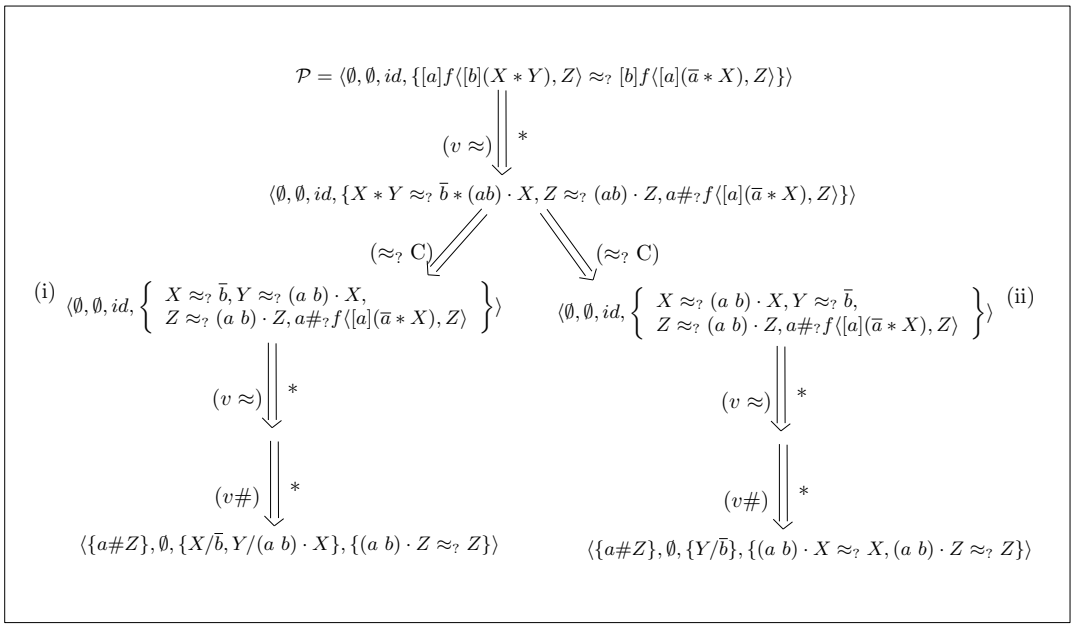


Figure 6. Derivation tree for nominal C-unification.

Example 17 (Nominal C-unification with $\mathscr{X}$ equals to the empty set)**.** This example exhibits the execution of the nominal C-unification algorithm for the initial problem

$$\mathscr{P} = \langle \emptyset,\ \emptyset,\ id,\ \{[a]f\langle[b](X * Y), Z\rangle \approx [b]f\langle[a](\overline{a} * X), Z\rangle\}\rangle,$$

where the set of protected variables ($\mathscr{X}$) is empty; thus, there exists no restriction over the variables of the problem. Notice that the application of rule $(\approx_? \mathbf{C})$ generates two branches that are represented by items (i) and (ii) in the example. The algorithm generates the leaves $\langle \{a\#Z\},\ \emptyset,\ \{X/\overline{b}, Y/(a\,b).X\},\ \{(a\,b).Z \approx_? Z\}\rangle$, and $\langle \{a\#Z\},\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z\}\rangle$. By Theorem 18, the union of the solutions of these two leaves is equal to the set of solutions of the initial problem $\mathscr{P}$. As shown in Ayala-Rincón et al. (2017), the complete set of solutions of $\langle \{a\#Z\},\ \emptyset,\ \{X/\overline{b}, Y/(a\,b).X\},\ \{(a\,b).Z \approx_? Z\}\rangle$ is unitary whereas the complete set of solutions of $\langle \{a\#Z\},\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z\}\rangle$ is infinite. Figure 6 illustrates the C-unification derivation tree for $\mathscr{P}$.

$$\begin{aligned} \mathscr{P} = {} & \langle \emptyset,\ \emptyset,\ id,\ \{[a]f\langle[b](X * Y), Z\rangle \approx [b]f\langle[a](\overline{a} * X), Z\rangle\}\rangle \\ \Rightarrow_{(\approx_?[\mathbf{ab}])} {} & \langle \emptyset,\ \emptyset,\ id,\ \{f\langle[b](X * Y), Z\rangle \approx_? f\langle[b](\overline{b} * (a\,b).X), (a\,b).Z\rangle,\ a\#_? f\langle[a](\overline{a} * X), Z\rangle\}\rangle \end{aligned}$$

$$\Rightarrow_{(\approx_?\mathbf{app})} \langle \emptyset,\ \emptyset,\ id,\ \{\langle [b](X*Y), Z\rangle \approx_? \langle [b](\overline{b}*(a\,b).X), (a\,b).Z\rangle,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{pair})} \langle \emptyset,\ \emptyset,\ id,\ \{[b](X*Y) \approx_? [b](\overline{b}*(a\,b).X),\ Z \approx_? (a\,b).Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{aa}])} \langle \emptyset,\ \emptyset,\ id,\ \{X*Y \approx_? (\overline{b}*(a\,b).X),\ Z \approx_? (a\,b).Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

(1)

$$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset,\ \emptyset,\ id,\ \{X \approx_? \overline{b},\ Y \approx_? (a\,b).X,\ Z \approx_? (a\,b).Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset,\ \emptyset,\ \{X/\overline{b}\},\ \{Y \approx_? (a\,b).X,\ Z \approx_? (a\,b).Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset,\ \emptyset,\ \{X/\overline{b},\ Y/(a\,b).X\},\ \{Z \approx_? (a\,b).Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \emptyset,\ \{X/\overline{b},\ Y/(a\,b).X\},\ \{(a\,b).Z \approx_? Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{app})} \langle \emptyset,\ \emptyset,\ \{X/\overline{b},\ Y/(a\,b).X\},\ \{(a\,b).Z \approx_? Z,\ a\#_? \langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{pair})} \langle \emptyset,\ \emptyset,\ \{X/\overline{b},\ Y/(a\,b).X\},\ \{(a\,b).Z \approx_? Z,\ a\#_? [a](\overline{a}*X),\ a\#_? Z\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{a}[\mathbf{a}])} \langle \emptyset,\ \emptyset,\ \{X/\overline{b},\ Y/(a\,b).X\},\ \{(a\,b).Z \approx_? Z,\ a\#_? Z\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{var})} \langle \{a\#Z\},\ \emptyset,\ \{X/\overline{b}, Y/(a\,b).X\},\ \{(a\,b).Z \approx_? Z\}\rangle$$

(2)

$$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset,\ \emptyset,\ id,\ \{X \approx_? (a\,b).X,\ Y \approx_? \overline{b},\ Z \approx_? (a\,b).Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \emptyset,\ id,\ \{(a\,b).X \approx_? X,\ Y \approx_? \overline{b},\ Z \approx_? (a\,b).Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset,\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_? X,\ Z \approx_? (a\,b).Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z,\ a\#_? f\langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{app})} \langle \emptyset,\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z,\ a\#_? \langle [a](\overline{a}*X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{pair})} \langle \emptyset,\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z,\ a\#_? [a](\overline{a}*X),\ a\#_? Z\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{a}[\mathbf{a}])} \langle \emptyset,\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z,\ a\#_? Z\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{var})} \langle \{a\#Z\},\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z\}\rangle$$

Theorem 18 (Correctness of $\Rightarrow_{\upsilon}^{*}$)**.** Let $\mathscr{P}$ be a valid quadruple.

(1) (Soundness of $\Rightarrow_{\upsilon}^{*}$) If $\mathscr{P} \Rightarrow_{\upsilon}^{*} \mathscr{Q}$, $\mathscr{Q}$ is a leaf and $\langle \nabla, \sigma\rangle \in \mathscr{U}_C(\mathscr{Q})$ then $\langle \nabla, \sigma\rangle \in \mathscr{U}_C(\mathscr{P})$.
(2) (Completeness of $\Rightarrow_{\upsilon}^{*}$) $\langle \nabla, \sigma\rangle \in \mathscr{U}_C(\mathscr{P})$ if and only if there exists a leaf $\mathscr{Q}$ such that $\mathscr{P} \Rightarrow_{\upsilon}^{*} \mathscr{Q}$ and $\langle \nabla, \sigma\rangle \in \mathscr{U}_C(\mathscr{Q})$.

Definition 19 (Proper problem)**.** A quadruple $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P\rangle$ is called a *proper problem* if every commutative function symbol in P has a pair as argument.

Theorem 20 (Characterisation of successful leaves)**.** Let $\mathscr{Q} = \langle \Delta, \mathscr{X}, \delta, Q\rangle$ be a leaf. If $\mathscr{Q}$ is a proper problem and there exists $\langle \nabla, \sigma\rangle \in \mathscr{U}_C(\mathscr{Q})$ with $\mathtt{dom}(\sigma) \cap \mathscr{X} = \emptyset$, then Q is a fixed point problem.

4. Nominal C-matching

In this section we restrict our attention to *nominal matching problems*: a nominal unification problem whose solutions should be applied only to the left-hand side of the nominal equations.

We specify a nominal C-matching algorithm that consists of applications of the *matching step* rules presented in Figure 7. These rules basically apply the rules in Figures 3, 4 and 5, but now the set $\mathscr{X}$ of protected variables plays an important role and should be defined as the variables occurring in the right-hand sides of the set of equational constraints P in the input problem.

Definition 21 (Protected variables and C-matching problems)**.** The set of protected variables for a matching problem $\langle \nabla, P\rangle$ (see Definition 9) is the set of *right-hand side variables* of the equational constraints in P, denoted by $\mathtt{Rvar}(P)$, i.e., $\mathtt{Rvar}(P) = \{X \mid s \approx_? t \in P \text{ and } X \in \mathtt{var}(t)\}$. The quadruple associated with the C-matching problem $\langle \nabla, P\rangle$ is given by $\langle \nabla, \mathtt{Rvar}(P), id, P\rangle$.

$$(\mu_\upsilon)\ \frac{\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}}{\mathscr{P} \Rightarrow_\mu \mathscr{Q}} \qquad\qquad (\mu_{\mathbf{fp}})\ \frac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{\pi.X \approx_? X\}\rangle}{\langle \nabla \cup \mathtt{dom}(\pi)\#X,\ \mathscr{X},\ \sigma,\ P\rangle}\ , P = P_{fp_\approx}$$

Figure 7. Matching step.

When solving a problem according to the rules in Figure 7 using the protected variables given in Definition 21, the domain of a substitution that is a solution will be disjoint from the set of right-hand side variables of the problem.

Derivation with rules of Figure 7 is denoted by $\Rightarrow_\mu$.

Definition 22 (Solution for a C-matching problem)**.** A *C-matching solution* for a quadruple $\mathscr{P}$ of the form $\langle \Delta, \mathtt{Rvar}(P), \delta, P\rangle$ is a pair $\langle \nabla, \sigma\rangle$, where $\mathtt{dom}(\sigma) \cap \mathtt{Rvar}(P) = \emptyset$, and the following conditions are satisfied:

(1) $\nabla \vdash \Delta\sigma$;
(2) $\nabla \vdash a \# t\sigma$, if $a\#_? t \in P$;
(3) $\nabla \vdash s\sigma \approx_{\{\alpha,C\}} t$, if $s \approx_? t \in P$;
(4) there is a substitution λ such that $\nabla \vdash \delta\lambda \approx \sigma$.

A *C-matching solution* for the problem $\langle \Delta, P\rangle$ is a solution for $\langle \Delta, \mathtt{Rvar}(P), id, P\rangle$, its associated C-matching problem. The *solution set* for a matching problem $\mathscr{P}$ is denoted by $\mathscr{M}_C(\mathscr{P})$.

Remark 23. We will call a quadruple $\mathscr{Q}$ a *matching leaf* if $\mathscr{Q}$ is a normal form w.r.t. $\Rightarrow_\mu$.

4.1 Auxiliary properties of nominal C-matching

We now present the main auxiliary lemmas related with nominal C-unification notions included in the formalisation.

Lemma 24 ($\mathscr{U}_C$ and $\mathscr{M}_C$ equivalence)**.** Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P\rangle$ be a quadruple. Then, $\langle \nabla, \sigma\rangle \in \mathscr{U}_C(\mathscr{P})$ if and only if $\langle \nabla, \sigma\rangle \in \mathscr{M}_C(\mathscr{P})$.

Proof. The formalisation follows straightforwardly from the definitions of $\mathscr{U}_C(\mathscr{P})$ and $\mathscr{M}_C(\mathscr{P})$. □

Lemma 25 (Preservation of $\mathtt{Rvar}$ by $\Rightarrow_\mu$)**.** Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P\rangle$ and $\mathscr{Q} = \langle \Delta', \mathscr{X}, \delta', Q\rangle$ such that $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$. Then $\mathtt{Rvar}(Q) \subseteq \mathtt{Rvar}(P)$.

Proof. The formalisation follows by case analysis on the $\Rightarrow_\mu$ reduction. □

Corollary 26 (Intersection emptiness preservation with right-hand side variables by $\Rightarrow_\mu$)**.** Let $\mathscr{P}$ and $\mathscr{Q}$ be two quadruples, $\langle \Delta, \mathscr{X}, \delta, P\rangle$ and $\langle \Delta', \mathscr{X}, \delta', Q\rangle$, respectively, and $\mathscr{Y}$ be an arbitrary set of variables. If $\mathtt{Rvar}(P) \cap \mathscr{Y} = \emptyset$ and $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$, then $\mathtt{Rvar}(Q) \cap \mathscr{Y} = \emptyset$.

Proof. This is indeed an easy set theoretically based corollary of Lemma 25. □

Corollary 27 (Preservation of valid quadruples by $\Rightarrow_\mu$)**.** If $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$ and $\mathscr{P}$ is valid then $\mathscr{Q}$ is also valid.

Proof. The formalisation follows from Lemma 14. □

Lemma 28 (Decidability of $\Rightarrow_\mu$)**.** For all quadruple $\mathscr{P}$ it is possible to decide whether there exists $\mathscr{Q}$ such that $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$. Thus, it is also possible to decide whether $\mathscr{P}$ is a leaf.

Proof. The formalisation is obtained by decidability of the relation $\Rightarrow_\upsilon$ (item (i) of Theorem 11.) □

4.2 Main formalised properties for nominal C-matching

Theorem 29 (Termination of $\Rightarrow_\mu$)**.** The relation $\Rightarrow_\mu$ is terminating.

Proof. The proof is by case analysis on the derivation rules of the relation $\Rightarrow_\mu$, and uses a lexicographic measure over sets of equation and freshness constraints. The measure is given by

$$\left\langle |\mathtt{var}(P)|,\ \sum_{s\approx_? t\in P} |s|+|t|,\ |P_\approx/P_{fp_\approx}|,\ \sum_{a\#_? s\in P} |s| \right\rangle$$

Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P\rangle$ and $\mathscr{Q} = \langle \nabla, \mathscr{X}, \sigma, Q\rangle$ such that $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$.
For the case of rule (μ_υ), Theorem 11 item (ii) is applied.
For the case of an application of rule $(\mu_{\mathbf{fp}})$, one observes that:

(1) $|\mathtt{var}(Q)| \leq |\mathtt{var}(P)|$,

(2) $\sum_{s\approx_? t\in Q} |s|+|t| < \sum_{s\approx_? t\in P} |s|+|t|$,

(3) $|Q_\approx/Q_{fp_\approx}| = |P_\approx/P_{fp_\approx}|$ and

(4) $\sum_{a\#_? s\in Q} |s| = \sum_{a\#_? s\in P} |s|$.

Therefore the measure also decreases in this case, which concludes the proof. □

Lemma 30 (Preservation of solutions by $\Rightarrow_\mu$)**.** Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P\rangle$ be a valid quadruple and $\mathscr{Q} = \langle \Delta', \mathscr{X}, \delta', Q\rangle$. If $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$, $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$ and $\langle \nabla, \sigma\rangle \in \mathscr{M}_C(\mathscr{Q})$, then $\langle \nabla, \sigma\rangle \in \mathscr{M}_C(\mathscr{P})$.

Proof. The proof is by case analysis on the derivation rules of $\Rightarrow_\mu$.

According Definition 22, one has $\mathtt{Rvar}(Q) \cap \mathtt{dom}(\sigma) = \emptyset$. From this, the hypothesis $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$ and using Lemmas 15 (item (i)) and 24 one concludes the case of rule (μ_υ).

For the case of rule $(\mu_{\mathbf{fp}})$, one needs to conclude the conditions of Definition 22 for the pair $\langle \nabla, \sigma \rangle$ w.r.t. $\mathscr{P}$. Condition (iv) is trivially satisfied. The first condition is proved just observing that every constraint $a\#X$ in Δ is also in $\Delta \cup \mathtt{dom}(\pi)$. The second condition is easily proved from the fact that if $a\#_? s \in P \uplus \{\pi.X \approx_? X\}$ then $a\#_? s \in P$. Then, one applies the hypothesis $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$ using Definition 22, item (ii), to conclude. The third condition is proved by analysis of two cases. The first case is when $s \approx_? t \in P \uplus \{\pi.X \approx_? X\}$ being the equation $s \approx_? t$ equal to $\pi.X \approx_? X$. In this case, one starts proving the statement $X \cap \mathtt{dom}(\sigma) = \emptyset$ using the hypothesis $\mathtt{Rvar}(P \uplus \{\pi.X \approx_? X\}) \cap \mathtt{dom}(\sigma) = \emptyset$. From this, $(\pi.X)\sigma$ can be replaced by $\pi.X$ in the objective $\nabla \vdash \pi X \sigma \approx_{\{\alpha,C\}} X$, remaining to prove that $\nabla \vdash \pi.X \approx_{\{\alpha,C\}} X$. Then, using the condition (i) of Definition 22 of hypothesis $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$ one has that $\nabla \vdash (\Delta \cup \mathtt{dom}(\pi)\#X)\sigma$. Since $X \notin \mathtt{dom}(\sigma)$, one concludes that $\mathtt{dom}(\pi)\#X \subseteq \nabla$ and then the objective is proved using the definition of $\approx_{\{\alpha,C\}}$ for the case of suspensions. The second case is when $s \approx_? t \in P$. This case is trivial, and uses hypothesis $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$ with Definition 22, item (iii). □

Theorem 31 (Completeness of $\Rightarrow_\mu$)**.** Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P \rangle$ a valid quadruple that is not a matching leaf, if $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$, then $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{P})$ if and only if there exists $\mathscr{Q}$ such that $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$.

Proof. Necessity is proved by case analysis on the derivation rules of $\Rightarrow_\mu$.

Lemma 28 is applied to the premise that $\mathscr{P}$ is not a matching leaf to obtain that there exists $\mathscr{Q}'$ such that $\mathscr{P} \Rightarrow_\mu \mathscr{Q}'$. Then for the case of rule (μ_υ), using Lemmas 16 (item (iii)) and 24 it is proved the assertion that there exists $\mathscr{Q}''$ such that $\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}''$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q}')$. From this, using again Lemma 24, applying rule (μ_υ) and using Corollary 26 one concludes.

For the case of rule $(\mu_{\mathbf{fp}})$, $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P' \uplus \{\pi.X \approx_? X\} \rangle$ with $P' = P'_{fp_\approx}$. The quadruple $\mathscr{Q} = \langle \Delta \cup \mathtt{dom}(\pi)\#X, \mathscr{X}, \delta, P \rangle$ will be a witness. Thus, $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$ follows by an application of rule $(\mu_{\mathbf{fp}})$. To prove that $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$, one has to show that the conditions of Definition 22 are satisfied, having as hypothesis that $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{P})$. Conditions (ii), (iii) and (iv) are trivially verified and intersection emptiness is proved using Corollary 26. For condition (i), a constraint $a\#X$ is chosen that is in $\Delta \cup \mathtt{dom}(\pi)\#X$ to analyse if it is either in Δ or $\mathtt{dom}(\pi)\#X$. If $a\#X$ is in Δ the proof is trivial, otherwise one first proves the assertion that $\{X\} \cap \mathtt{dom}(\sigma) = \emptyset$ from the hypotheses that $\mathscr{P}$ is valid and $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$. This allows to replace every $X\sigma$ and every $(\pi.X)\sigma$, respectively, just by X and $\pi.X$, because $X \notin \mathtt{dom}(\sigma)$. Since $\pi.X \approx_? X$ is in $P \uplus \{\pi.X \approx_? X\}$ and $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{P})$, we have that $\nabla \vdash (\pi.X)\sigma \approx_{\{\alpha,C\}} X$, therefore $\nabla \vdash \pi.X \approx_{\{\alpha,C\}} X$ and then $\mathtt{dom}(\pi)\#X \subseteq \nabla$. On the other hand, having $a \in \mathtt{dom}(\pi)$ as hypothesis, one has to prove that $\nabla \vdash a\#X\sigma$, which is the same as $\nabla \vdash a\#X$. Using the fact that $\mathtt{dom}(\pi)\#X \subseteq \nabla$, one concludes.

Sufficiency is formalised as a direct consequence of Lemma 30. □

Theorem 32 (Soundness of $\Rightarrow_\mu^*$)**.** Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P \rangle$ be a valid quadruple and $\mathscr{P} \Rightarrow_\mu^* \mathscr{Q}$. If $\mathscr{Q}$ is a matching leaf and $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$ such that $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$ then $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{P})$.

Proof. The proof uses Corollaries 27 and 26 and Lemma 30, and it is done by induction on the number of steps of $\Rightarrow_\mu$. If $\mathscr{P} = \mathscr{Q}$ the proof is trivial. In the case where $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$, Lemma 30 is applied to conclude. When $\mathscr{P} \Rightarrow_\mu \mathscr{R}$ and $\mathscr{R} \Rightarrow_\mu^* \mathscr{Q}$, one uses Lemma 30, IH and Lemmas 27 and 26 to conclude. □

Theorem 33 (Completeness of $\Rightarrow_\mu^*$)**.** Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P \rangle$ be a valid quadruple and $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{P})$. Then there exists a matching leaf $\mathscr{Q}$ such that $\mathscr{P} \Rightarrow_\mu^* \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$.

Proof. The formalisation follows by well-founded induction on the number of applications of $\Rightarrow_\mu$. Also, Lemma 28 is applied in the analysis of the cases where either $\mathscr{P}$ is a matching leaf or there

exists Q' such that $\mathscr{P} \Rightarrow_\mu \mathscr{Q}'$. If $\mathscr{P}$ is a matching leaf then $\mathscr{P} = \mathscr{Q}$ and the proof is completed. If there exists Q' such that $\mathscr{P} \Rightarrow_\mu \mathscr{Q}'$, one applies Lemma 28 to obtain that $\mathscr{P}$ is not a matching leaf. Lemma 31 is applied to the premise that $\mathscr{P}$ is not a matching leaf. From this and the hypothesis $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{P})$ one obtains that there exists $\mathscr{Q}'$ such that $\mathscr{P} \Rightarrow_\mu \mathscr{Q}'$.

The IH is established as the following statement: $\forall \mathscr{R}$ valid, if $\mathscr{P} \Rightarrow_\mu \mathscr{R}$, $\mathtt{Rvar}(R) \cap \mathtt{dom}(\sigma)$ and $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{R})$, then there exists $\mathscr{S}$, such that $\mathscr{R} \Rightarrow_\mu^* \mathscr{S}$ and $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{S})$. This is applied to the hypothesis $\mathscr{P} \Rightarrow_\mu \mathscr{Q}'$ to conclude that there exists $\mathscr{Q}$, such that $\mathscr{Q}' \Rightarrow_\mu^* \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q}')$. The other premises of IH are achieved with the auxiliary results given by Lemma 27 and Corollary 26. Finally, by case analysis on the statement $\mathscr{Q}' \Rightarrow_\mu^* \mathscr{Q}$, one concludes. □

Theorem 34 (Characterisation of successful matching leaves)**.** Let $\mathscr{Q} = \langle \Delta, \mathscr{X}, \delta, Q \rangle$ a matching leaf, if $\mathscr{Q}$ is a proper problem and there exists $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$, then $Q = \emptyset$.

Proof. First one proves the assertion that Q is a fixed point problem. This statement is proved using Theorem 20, Lemma 24 and rule (μ_υ) of the definition of matching step (Figure 7). Therefore, if Q is a fixed problem it must be equal to the empty set, otherwise $\mathscr{Q}$ could be reduced by an application of rule $(\mu_{\mathbf{fp}})$ of Figure 7, which contradicts the fact that $\mathscr{Q}$ is a matching leaf. □

Example 35 (Nominal C-matching)**.** This example is similar to Example 17, but now the set of protected variables is equal to the right-hand side variables of the initial problem, that is $\{X, Z\}$. This results in the execution of the nominal C-matching algorithm that provides the matching leaves

$$\langle \emptyset,\ \{X, Z\},\ \{\ Y/(a\ b).X\},\ \{X \approx_? \overline{b},\ (a\ b).Z \approx_? Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle,$$

$$\text{and, } \langle \{a\#X, b\#X, a\#Z, b\#Z\},\ \{X, Z\},\ \{Y/\overline{b}\},\ \emptyset\rangle.$$

Since X is a protect variable, the former problem has no solution due the fact that the equation $X \approx_? \overline{b}$ cannot be solved since X cannot be instantiated. The latter problem has just one solution given by $\langle \{a\#X, b\#X, a\#Z, b\#Z\}, \{Y/\overline{b}\}\rangle$. Theorems 32 and 33 show that this solution is the unique C-matching solution for the initial problem.

$$\begin{aligned}
\mathscr{P} = {} & \langle \emptyset,\ \{X, Z\},\ id,\ \{[a]f\langle [b](X * Y), Z\rangle \approx [b]f\langle [a]((\overline{a} * X)), Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?[\mathbf{ab}])} & \langle \emptyset,\ \{X, Z\},\ id,\ \{f\langle [b](X * Y), Z\rangle \approx_? f\langle [b](\overline{b} * (a\ b).X), (a\ b).Z\rangle,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{app})} & \langle \emptyset,\ \{X, Z\},\ id,\ \{\langle [b](X * Y), Z\rangle \approx_? \langle [b](\overline{b} * (a\ b).X), (a\ b).Z\rangle,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{pair})} & \langle \emptyset,\ \{X, Z\},\ id,\ \{[b](X * Y) \approx_? [b](\overline{b} * (a\ b).X),\ Z \approx_? (a\ b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?[\mathbf{aa}])} & \langle \emptyset,\ \{X, Z\},\ id,\ \{X * Y \approx_? (\overline{b} * (a\ b).X),\ Z \approx_? (a\ b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle
\end{aligned}$$

(1)

$$\begin{aligned}
\Rightarrow_{(\approx_?\mathbf{C})} & \langle \emptyset,\ \{X, Z\},\ id,\ \{X \approx_? \overline{b},\ Y \approx_? (a\ b).X,\ Z \approx_? (a\ b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{inst})} & \langle \emptyset,\ \{X, Z\},\ \{Y/(a\ b).X\},\ \{X \approx_? \overline{b},\ Z \approx_? (a\ b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{inv})} & \langle \emptyset,\ \{X, Z\},\ \{\ Y/(a\ b).X\},\ \{X \approx_? \overline{b},\ (a\ b).Z \approx_? Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle
\end{aligned}$$

(2)

$$\begin{aligned}
\Rightarrow_{(\approx_?\mathbf{C})} & \langle \emptyset,\ \{X, Z\},\ id,\ \{X \approx_? (a\ b).X,\ Y \approx_? \overline{b},\ Z \approx_? (a\ b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{inv})} & \langle \emptyset,\ \{X, Z\},\ id,\ \{(a\ b).X \approx_? X,\ Y \approx_? \overline{b},\ Z \approx_? (a\ b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle
\end{aligned}$$

$$\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset,\ \{X,Z\},\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_?^{} X,\ Z \approx_?^{} (a\,b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X,Z\},\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_?^{} X,\ (a\,b).Z \approx_?^{} Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{app})} \langle \emptyset,\ \{X,Z\},\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_?^{} X,\ (a\,b).Z \approx_?^{} Z,\ a\#_? \langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{pair})} \langle \emptyset,\ \{X,Z\},\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_?^{} X,\ (a\,b).Z \approx_?^{} Z,\ a\#_? [a](\overline{a} * X),\ a\#_? Z\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{a[a]})} \langle \emptyset,\ \{X,Z\},\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_?^{} X,\ (a\,b).Z \approx_?^{} Z,\ a\#_? Z\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{var})} \langle \{a\#Z\},\ \{X,Z\},\ \{Y/\overline{b}\},\ \{(a\,b).X \approx_?^{} X,\ (a\,b).Z \approx_?^{} Z\}\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X, a\#Z\},\ \{X,Z\},\ \{Y/\overline{b}\},\ \{(a\,b).Z \approx_?^{} Z\}\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X, a\#Z, b\#Z\},\ \{X,Z\},\ \{Y/\overline{b}\},\ \emptyset\rangle$$

Example 36 (Nominal C-equivalence checking)**.** This example exhibits the execution of the nominal C-unification algorithm applied to nominal C-equivalence check. In item (a), the set of protected variables, $\{X,Y,Z\}$, consists now of all variables in the input problem. The algorithm generates two leaves

$$\langle \emptyset,\ \{X,Y,Z\},\ id,\ \{X \approx_?^{} \overline{b},\ Y \approx_?^{} (a\,b).X, (a\,b).Z \approx_?^{} Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

and, $\langle \emptyset,\ \{X,Y,Z\},\ id,\ \{(a\,b).X \approx_?^{} X,\ Y \approx_?^{} \overline{b},\ (a\,b).Z \approx_?^{} Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$.

Both are quadruples that have equations without solutions. In the former one, the X cannot be instantiated to solve $X \approx_?^{} \overline{b}$, and in the latter one, Y cannot be instantiated to solve $Y \approx_?^{} \overline{b}$.

In item (b), the set of protected variables, $\{X,Y\}$, consists also of all variables in the input problem, but the generated leaves are

$$\langle \emptyset,\ \{X,Y\},\ id,\ \{X \approx_?^{} \overline{b},\ \overline{b} \approx_?^{} (a\,b).X, (a\,b).Y \approx_?^{} Y,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$$

and, $\langle \{a\#X, b\#X, a\#Y, b\#Y\},\ \{X,Y\},\ id,\ \emptyset\rangle$

The first leaf has also equations with the protected variable X. Namely, in equations $X \approx_?^{} \overline{b}$ and $\overline{b} \approx_?^{} (a\,b).X$ X cannot be instantiated. Thus, neither equation has solutions. On the other branch, the second leaf provides a solution given by the freshness context $\{a\#X, b\#X, a\#Y, b\#Y\}$.

(a) $\langle \emptyset,\ \{X,Y,Z\},\ id,\ \{[a]f\langle [b](X * Y), Z\rangle \approx [b]f\langle [a](\overline{a} * X), Z\rangle\}\rangle$

$$\Rightarrow_{(\approx_?[\mathbf{ab}])} \langle \emptyset,\ \{X,Y,Z\},\ id,\ \{f\langle [b](X * Y), Z\rangle \approx_?^{} f\langle [b](\overline{b} * (a\,b).X), (a\,b).Z\rangle,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{app})} \langle \emptyset,\ \{X,Y,Z\},\ id,\ \{\langle [b](X * Y), Z\rangle \approx_?^{} \langle [b](\overline{b} * (a\,b).X), (a\,b).Z\rangle,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{pair})} \langle \emptyset,\ \{X,Y,Z\},\ id,\ \{[b](X * Y) \approx_?^{} [b](\overline{b} * (a\,b).X),\ Z \approx_?^{} (a\,b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{aa}])} \langle \emptyset,\ \{X,Y,Z\},\ id,\ \{X * Y \approx_?^{} (\overline{b} * (a\,b).X),\ Z \approx_?^{} (a\,b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

(1)

$$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset,\ \{X,Y,Z\},\ id,\ \{X \approx_?^{} \overline{b},\ Y \approx_?^{} (a\,b).X,\ Z \approx_?^{} (a\,b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X,Y,Z\},\ id,\ \{X \approx_?^{} \overline{b},\ Y \approx_?^{} (a\,b).X, (a\,b).Z \approx_?^{} Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

(2)

$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset,\ \{X,Y,Z\},\ id,\ \{X \approx_? (a\ b).X,\ Y \approx_? \overline{b},\ Z \approx_? (a\ b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X,Y,Z\},\ id,\ \{(a\ b).X \approx_? X,\ Y \approx_? \overline{b},\ Z \approx_? (a\ b).Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X,Y,Z\},\ id,\ \{(a\ b).X \approx_? X,\ Y \approx_? \overline{b},\ (a\ b).Z \approx_? Z,\ a\#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$

(b) $\langle \emptyset,\ \{X,Y\},\ id,\ \{[a]f\langle [b](X * \overline{b}), Y\rangle \approx [b]f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?[\mathbf{ab}])} \langle \emptyset,\ \{X,Y\},\ id,\ \{f\langle [b](X * \overline{b}), Y\rangle \approx_? f\langle [b](\overline{b} * (a\ b).X), (a\ b).Y\rangle,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{app})} \langle \emptyset,\ \{X,Y\},\ id,\ \{\langle [b](X * \overline{b}), Y\rangle \approx_? \langle [b](\overline{b} * (a\ b).X), (a\ b).Y\rangle,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{pair})} \langle \emptyset,\ \{X,Y\},\ id,\ \{[b](X * \overline{b}) \approx_? [b](\overline{b} * (a\ b).X),\ Y \approx_? (a\ b).Y,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?[\mathbf{aa}])} \langle \emptyset,\ \{X,Y\},\ id,\ \{X * \overline{b} \approx_? (\overline{b} * (a\ b).X),\ Y \approx_? (a\ b).Y,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

(1)

$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset,\ \{X,Y\},\ id,\ \{X \approx_? \overline{b},\ \overline{b} \approx_? (a\ b).X,\ Y \approx_? (a\ b).Y,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X,Y\},\ id,\ \{X \approx_? \overline{b},\ \overline{b} \approx_? (a\ b).X, (a\ b).Y \approx_? Y,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

(2)

$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset,\ \{X,Y\},\ id,\ \{X \approx_? (a\ b).X,\ \overline{b} \approx_? \overline{b},\ Y \approx_? (a\ b).Y,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X,Y\},\ id,\ \{(a\ b).X \approx_? X,\ \overline{b} \approx_? \overline{b},\ Y \approx_? (a\ b).Y,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{refl})} \langle \emptyset,\ \{X,Y\},\ id,\ \{(a\ b).X \approx_? X,\ Y \approx_? (a\ b).Y,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X,Y\},\ id,\ \{(a\ b).X \approx_? X,\ (a\ b).Y \approx_? Y,\ a\#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{app})} \langle \emptyset,\ \{X,Y\},\ id,\ \{(a\ b).X \approx_? X,\ (a\ b).Y \approx_? Y,\ a\#_? \langle [a](\overline{a} * X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{pair})} \langle \emptyset,\ \{X,Y\},\ id,\ \{(a\ b).X \approx_? X,\ (a\ b).Y \approx_? Y,\ a\#_? [a](\overline{a} * X),\ a\#_? Y\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{a}[\mathbf{a}])} \langle \emptyset,\ \{X,Y\},\ id,\ \{(a\ b).X \approx_? X,\ (a\ b).Y \approx_? Y,\ a\#_? Y\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{var})} \langle \{a\#Y\},\ \{X,Y\},\ id,\ \{(a\ b).X \approx_? X,\ (a\ b).Y \approx_? Y\}\rangle$

$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X, a\#Y\},\ \{X,Y\},\ id,\ \{(a\ b).Y \approx_? Y\}\rangle$

$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X, a\#Y, b\#Y\},\ \{X,Y\},\ id,\ \emptyset\rangle$

5. Adapting the recursive nominal C-unification algorithm to handle protected variables

In this section we present a recursive matching algorithm obtained by adapting the algorithm presented by Ayala-Rincón et al. (2019), and discuss the most interesting aspects of the adaptation.

The functional algorithm for nominal C-unification let us unify two terms t and s. By using the appropriate set of protected variables, the algorithm can be adapted to do C-matching and C-equality checking. The algorithm is recursive (see Algorithm 1) and keeps track of the protected variables, the current context, the substitutions done so far, the remaining terms left to unify and the current fixed point equations. Therefore, the algorithm receives as input a quintuple $(\mathscr{X}, \Delta, \sigma, PrbLst, FPEqLst)$, where $\mathscr{X}$ is the set of protected variables, Δ is the context we are working with, σ represents the substitutions already made, $PrbLst$ is a list of equations we must

still solve (each equation $t \approx_? s$ is represented as a pair (t, s) in Algorithm 1) and $FPEqLst$ is a list of fixed point equations we have already computed.

Remark 37. In contrast with Ayala-Rincón et al. (2018*a*), Algorithm 1 has an extra parameter to store fixed point equations. This let us test termination of the algorithm just by checking if *PrbLst* is empty.

The first call to the algorithm in order to unify the terms t and s is simply: $\texttt{UNIFY}(\emptyset, \emptyset, id, [(t, s)], \emptyset)$. The algorithm eventually terminates, returning a list (possibly empty) of triples of the form $(\Delta, \sigma, FPEqLst)$. These triples correspond to the leafs of the algorithm of Ayala-Rincón et al. (2018*a*).

Although long, the algorithm is simple. It starts by analysing the list of terms it needs to unify. If *PrbLst* is an empty list, then it has finished and can return the answer computed so far, which is the list: $[(\Delta, \sigma, FPEqLst)]$. If *PrbLst* is not empty, then there are terms to unify, and the algorithm starts by trying to unify the terms t and s in the head of the list. The algorithm calls itself on progressively simpler versions of the problem until it finishes.

5.1 Main algorithm and modifications made

The pseudocode for the algorithm is presented in Algorithm 1. In relation to the algorithm presented in Ayala-Rincón et al. (2019), the only changes are: checking, in lines 6, 24 and 25 whether the variable X is in $\mathscr{X}$ or not, and the addition of the parameter $\mathscr{X}$ for a set of protected variables, which remains constant in the execution of the algorithm.

5.2 Auxiliary functions

Following the approach of Ayala-Rincón et al. (2016), freshness constraints are handled by auxiliary functions, making the main function `UNIFY` smaller. To deal with the freshness constraints, the following auxiliary functions, which come from Ayala-Rincón et al. (2016), were used:

- `fresh_subs?`(σ, Δ) returns the minimal context (Δ' in Algorithm 1) in which $a\#_? X\sigma$ holds, for every $a\#X$ in the context Δ.
- `fresh?`(a, t) computes and returns the minimal context (Δ' in Algorithm 1) in which a is fresh in t.

The two functions also return a Boolean (*bool*1 in Algorithm 1), to indicate if it was possible to find the mentioned context.

5.3 Interesting points on adapting the algorithm to handle protected variables

After the addition of protected variables, the proofs of soundness and completeness remained essentially the same as their previous versions, which were described in Ayala-Rincón et al. (2019). The interesting points in this regard are in Remarks 38 and 39.

Remark 38. In a preliminary attempt to extend the algorithm to handle protected variables, we considered reusing the previous code but checking, before instantiating a moderated variable $\pi.X$, whether X was in $\mathscr{X}$ or not. If it was, and the other term was not also of the form $\pi.X$ we would return an empty list of solutions, as X cannot be instantiated. If X was not in $\mathscr{X}$, the algorithm would ran as before. This naive approach fails. Consider, for instance, the case where you are trying to unify $\pi.X$ with $\pi'.Y$ and X is a protected variable but Y is not. In this case, the

Algorithm 1 - **First Part** - Functional Nominal C-Unification

```
1:  procedure UNIFY(𝒳, Δ, σ, PrbLst, FPEqLst)
2:      if nil?(PrbLst) then
3:          return list((Δ, σ, FPEqLst))
4:      else
5:          cons((t, s), PrbLst′) = PrbLst
6:          if (s matches π.X) and (X not in t) and (X not in 𝒳) then
7:              σ′ = {X/π^{-1} · t}
8:              σ″ = σ′ ∘ σ
9:              (Δ′, bool1) = fresh_subs?(σ′, Δ)
10:             Δ″ = Δ ∪ Δ′
11:             PrbLst″ = append((PrbLst′)σ′, (FPEqLst)σ′)
12:             if bool1 then return UNIFY(𝒳, Δ″, σ″, PrbLst″, nil)
13:             else return nil
14:             end if
15:         else
16:             if t matches ā then
17:                 if s matches ā then
18:                     return UNIFY(𝒳, Δ, σ, PrbLst′, FPEqLst)
19:                 else
20:                     return nil
21:                 end if
22:             else if t matches π.X then
23:                 if X not in s then
24:                     if X not in 𝒳 then
25:                         return nil
26:                     else
27:                                          ▷ Similar to case above where s is a suspension
28:                     end if
29:                 else if (s matches π′.X) then
30:                     FPEqLst′ = FPEqLst ∪ { π · X ≈_α π′ · X}
31:                     return UNIFY(𝒳, Δ, σ, PrbLst′, FPEqLst′)
32:                 else return nil
33:                 end if
34:             else if t matches ⟨⟩ then
35:                 if s matches ⟨⟩ then
36:                     return UNIFY(𝒳, Δ, σ, PrbLst′, FPEqLst)
37:                 else return nil
38:                 end if
39:             else if t matches ⟨t1, t2⟩ then
40:                 if s matches ⟨s1, s2⟩ then
41:                     PrbLst″ = cons((s1, t1), cons((s2, t2), PrbLst′))
42:                     return UNIFY(𝒳, Δ, σ, PrbLst″, FPEqLst)
43:                 else return nil
44:                 end if
45:             else if t matches [a]t1 then
46:                 if s matches [a]s1 then
47:                     PrbLst″ = cons((t1, s1), PrbLst′)
48:                     return UNIFY(𝒳, Δ, σ, PrbLst″, FPEqLst)
49:                 else if s matches [b]s1 then
50:                     (Δ′, bool1) = fresh?(a, s1)
51:                     Δ″ = Δ ∪ Δ′
52:                     PrbLst″ = cons((t1,  (a b) s1), PrbLst′)
53:                     if bool1 then
54:                         return UNIFY(𝒳, Δ″, σ, PrbLst″, FPEqLst)
55:                     else return nil
56:                     end if
57:                 else return nil
58:                 end if
```

Algorithm 1 - **Second Part** - Functional Nominal C-Unification

else if t `matches` $f\ t_1$ **then** ▷ f is not commutative
  if s `matches` $f\ s_1$ **then**
    $PrbLst'' = cons((t_1, s_1), PrbLst')$
    return UNIFY$(\mathscr{X}, \Delta, \sigma, PrbLst'', FPEqLst)$
  else return nil
  end if
else ▷ t is of the form $f^C(t_1, t_2)$
  if s `matches` $f^C(s_1, s_2)$ **then**
    $PrbLst_1 = cons((s_1, t_1), cons((s_2, t_2), PrbLst'))$
    sol_1 = UNIFY$(\mathscr{X}, \Delta, \sigma, PrbLst_1, FPEqLst)$
    $PrbLst_2 = cons((s_1, t_2), cons((s_2, t_1), PrbLst'))$
    sol_2 = UNIFY$(\mathscr{X}, \Delta, \sigma, PrbLst_2, FPEqLst)$
    return APPEND(sol_1, sol_2)
  else return nil
  end if
end if
end if
end if
end procedure

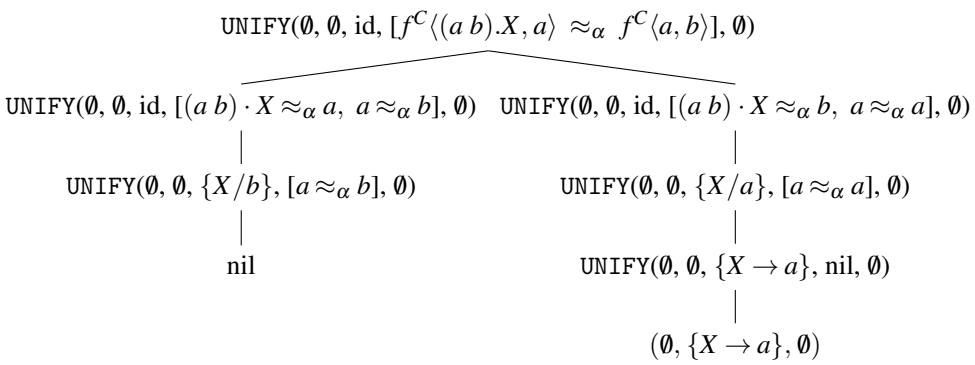
instantiation $X/(\pi^{-1} \oplus \pi').Y$ is not possible, but the algorithm should not return an empty list of solutions as the instantiation $Y/(\pi'^{-1} \oplus \pi).X$ is feasible.

Remark 39. The theorems of soundness and completeness of the algorithm had to be specified again, as the algorithm now has a new parameter $\mathscr{X}$ for the protected variables. In their new specification, it is not possible to plug in $Rvar(PrbLst)$ (where $Rvar(PrbLst)$ is the set of variables occurring in the right-hand side of the list of unification problems) as the set of protected variables $\mathscr{X}$ directly since the proofs of correctness and completeness are done by induction and from one recursive call of the algorithm to another the set $Rvar(PrbLst)$ may change, while $\mathscr{X}$ remains unchanged. The correct way to proceed is to prove the soundness and completeness of the algorithm with an arbitrary set of protected variables $\mathscr{X}$ and then, by a suitable choice of $\mathscr{X}$, obtain as corollaries the correctness of the algorithm for unification and matching.

5.4 Example of the algorithm

Example 40 illustrates the execution of the algorithm.

Example 40. This example shows how the algorithm proceeds in order to unify $f^C\langle(a\ b).X, a\rangle$ and $f^C\langle a, b\rangle$. Notice we have $\mathscr{X} = \emptyset$ in all calls to the function `UNIFY`.



6. Experiments comparing implementations

PVSIO is a PVS package that extends the ground evaluator with a predefined library of imperative programming languages features, among them input and output operators [Muñoz and Butler (2003)]. For our purposes, this means that we can run the formalised PVS function that performs unification with the help of the ground evaluator, and use the input and output capabilities provided by PVSIO to perform meaningful experiments.

Experiments to compare the Python 3 implementation with the executable code that can be run inside PVS via the PVSIO feature have been made. In the next sections, we describe the methodology used and present and discuss the results obtained so far.

6.1 Methodology

To compare the Python and the PVS implementation, we generated a fixed number of terms t and s to be unified and ran the implementations, measuring the time. By printing the Python results in the same way as the PVS implementation prints, it was possible to check whether the implementations match or not and how long each one took to unify. The results can be seen in Table 4.

Remark 41. The machine that ran the experiments has the following specifications:

- Operating System - MacOS High Sierra
- Processor - 3,6GHz Intel Core i7
- Memory - 16GB 2400 MHz DDR4
- Graphics - Radeon Pro 560 4096 MB

We generate the term t randomly according to the probabilities presented in Table 1. The number of different atoms, variables, function symbols and commutative function symbols is shown in Table 2. Finally, to generate a permutation, we first define a probability p ($0 \leq p \leq 1$) of generating a new swapping. Then, we generate a random number n between 0 and 1. If $n > p$ we stop generating swappings and return the permutation (i.e., the list of swappings) constructed so far. If $n \leq p$ we generate a new swapping, add this swapping to our current list of swappings and go back to generating a new random number n. We repeat the procedure until we fall in the case $n > p$. We used $p = 0.5$ in our experiments.

Finally, we generate the term s as a "copy with modifications" of the term t. These modifications are:

Table 1. Probability of generating each type of term.

Type of term	Probability
Atom	0.1
Moderated Variable	0.2
Unit	0.1
Abstraction	0.2
Pair	0.1
Function application	0.2
Commutative function application	0.1

Table 2. Number of different atoms, variables, function symbols and commutative function symbols.

Type of term	Number of different terms in our domain
Atom	10
Variable	10
Function symbols	5
Commutative function symbols	5

Table 3. Probability of making modifications in the term s when constructing it from the term t.

Type of modification	Probability
p_{var}	0.05
p_C	0.5
p_{abs}	0.5
p_{atom}	0.1

- With probability p_{var} we substitute part of the term t by a random moderated variable.
- If we encounter a commutative function application in t, with probability p_C we change the order of the two arguments.
- If we encounter an abstraction, with probability p_{abs} we correctly change the atom being abstracted (for instance, change a term $[a]t'$ to a term $[b](a\ b)\cdot t'$).
- If we encounter an atom, with probability p_{atom}, we change the atom.

The probabilities of doing these modifications are shown in Table 3.

Remark 42. Notice that if we encounter an atom in the term t and change it when constructing the term s this may result in non unifiable terms t and s. This is precisely what we hoped to accomplish, since we also want to test how the algorithm runs when the terms are not unifiable.

Table 4. Time each implementation took to unify a given number of terms.

Number of unification problems	Python - time	PVS - time
1000	< 1s	43s
2000	< 1s	1min24s
10000	3s	Error - stack overflow

6.2 Results and interpretation

First we checked if both implementations gave equal results, which was indeed the case. Next, we measured the running time, according to the number of terms being unified. The results are shown in Table 4.

The results show that the Python implementation is faster (which was expected since the PVS implementation is not LISP executing directly, but LISP executing under the PVS environment, which adds an overhead). What was not expected was the difference in the performance of the two implementations. Other surprising event was the PVS implementation giving an error when the number of unification problems grew to 10000.

7. Conclusion and Future Work

This paper presents an extension of the nominal C-unification algorithm proposed in Ayala-Rincón et al. (2018*a*), which permits the use of *protected variables*. When the set of protected variables is the set of variables in the right-hand side of nominal equational problems given as input, the algorithm outputs a nominal C-matcher for the input problem if one exists. If all the variables of a nominal unification problem are protected, the algorithm becomes a nominal C-equality checker. The nominal C-matching algorithm was checked through a formalisation in Coq which reused a formalisation of the unification algorithm in Ayala-Rincón et al. (2018*a*) plus additional formalisations related with the main desired properties of the C-matching algorithm that are termination, soundness and completeness.

This paper also extends the functional nominal C-unification algorithm of Ayala-Rincón et al. (2019), by adding a parameter for the set of protected variables. Moreover, experiments comparing the Python implementation with the executable code generated by PVS are performed and the results are discussed.

We are currently investigating why the PVS performance was significantly slower than the Python implementation and if the PVS implementation systematically fails when dealing with a large number of unification problems. We also plan on refining our experiment, by generating a fixed amount of unification problems (in our case 1000, 2000 and 10000 unification problems) a multiple number of times (instead of only one) and calculating the average of time that each implementation takes. To enhance the experiments, one could devise a recursive algorithm from the inductive set of rules of Ayala-Rincón et al. (2018*a*) and prove its correctness and completeness in Coq. This could be done by giving a heuristic on how to apply the rules (notice that the rules are non-deterministic and for a given $\mathscr{P}$ there may be more than one applicable reduction rule). Then, using the Coq feature of code extraction we would obtain executable code and we would be able to compare the Python implementation with this other implementation. With the Coq code extraction feature, we can obtain code in OCaml or Haskell that runs independently of the Coq environment. Therefore, we expect the code extracted this way to have a better performance than the PVS implementation and a competitive performance in relation to the Python implementation, although this needs to be verified with careful experiments. Other possible paths of future work include

investigating the formalisation of nominal AC-unification and matching, dealing with restricted cases such as linear AC-matching and working with unification modulo more general theories.

References

Ayala-Rincón, M., Carvalho-Segundo, W., Fernández, M., and Nantes-Sobrinho, D. 2017. *On Solving Nominal Fixpoint Equations.* In *Proc. of the 11th Int. Symp. on Frontiers of Combining Systems (FroCoS)*, volume 10483 of *LNCS*, pp. 209–226. Springer.

Ayala-Rincón, M., Carvalho-Segundo, W., Fernández, M., and Nantes-Sobrinho, D. 2018a. *Nominal C-Unification.* In *Post-proc. of the 27th Int. Symp. Logic-based Program Synthesis and Transformation (LOPSTR 2017)*, volume 10855 of *LNCS*, pp. 235–251. Springer.

Ayala-Rincón, M., de Carvalho-Segundo, W., Fernández, M., Nantes-Sobrinho, D., and Rocha-Oliveira, A. 2019. A Formalisation of Nominal alpha-equivalence with A, C, and AC Function Symbols. *Theor. Comput. Sci.*, 781:3–23.

Ayala-Rincón, M., Fernández, M., and Nantes-Sobrinho., D. 2016. *Nominal Narrowing.* In *Proc. of the 1st Int. Conf. on Formal Structures for Computation and Deduction (FSCD)*, volume 52 of *LIPIcs*, pp. 11:1–11:17.

Ayala-Rincón, M., Fernández, M., and Nantes-Sobrinho, D. 2018b. *Fixed Point Constraints for Nominal Equational Unfication.* In *Proc. of the 3rd Int. Conf. Formal Structures for Computation and Deduction (FSCD)*, volume 108 of *LIPIcs*, pp. 7:1–7:16.

Ayala-Rincón, M., Fernández, M., and Rocha-oliveira, A. C. 2016. *Completeness in PVS of a Nominal Unification Algorithm. ENTCS*, 323:57–74.

Ayala-Rincón, M., Fernández, M., Silva, G., and Nantes-Sobrinho, D. 2019. *A Certified Functional Nominal C-Unification Algorithm.* In *Pre-proc. of the 29th Int. Symp. Logic-based Program Synthesis and Transformation (LOPSTR)*.

Ayala-Rincón, M., de Carvalho-Segundo, W., Fernández, M., and Nantes-Sobrinho, D. 2019. A Formalisation of Nominal C-Matching through Unification with Protected Variables. *ENTCS*, 344:47 – 65.

Baader, F. 1986. The Theory of Idempotent Semigroups is of Unification Type Zero. *J. of Autom. Reasoning*, 2(3):283–286.

Baader, F. and Schulz, K. U. 1996. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. *J. of Sym. Computation*, 21(2):211–243.

Baader, F. and Snyder, W. 2001. Unification Theory. In *Handbook of Automated Reasoning (in 2 volumes)*, pp. 445–532. Elsevier and MIT Press.

Calvès, C. F. 2010. *Complexity and implementation of nominal algorithms*. PhD Thesis, King's College London.

Calvès, C. F. and Fernández, M. 2011. *The First-order Nominal Link.* In *Proc. of the 20th Int. Symp. Logic-based Program Synthesis and Transformation (LOPSTR)*, volume 6564 of *LNCS*, pp. 234–248. Springer.

Contejean, E. 2004. *A Certified AC Matching Algorithm.* In *Proc. of the 15th Int. Conf. on Rewriting Techniques and Applications, (RTA)*, volume 3091 of *LNCS*, pp. 70–84. Springer.

Fages, F. 1987. Associative-Commutative Unification. *J. of Sym. Computation*, 3:257–275.

Kapur, D. and Narendran, P. 1986. NP-Completeness of the Set Unification and Matching Problems. In *8th International Conference on Automated Deduction (CADE)*, volume 230 of *LNCS*, pp. 489–495. Springer.

Kapur, D. and Narendran, P. 1987. *Matching, Unification and Complexity. SIGSAM Bulletin*, 21(4):6–9.

Kapur, D. and Narendran, P. 1992. Complexity of Unification Problems with Associative-Commutative Operators. *J. of Autom. Reasoning*, 9(2):261–288.

Levy, J. and Villaret, M. 2010. *An Efficient Nominal Unification Algorithm.* In *Proc. of the 21st Int. Conf. on Rewriting Techniques and Applications (RTA)*, volume 6 of *LIPIcs*, pp. 209–226.

Muñoz, C. and Butler, R. 2003. Rapid prototyping in PVS. Technical Report NASA/CR-2003-212418, NIA-2003-03, NASA Langley Research Center (NIA).

Pitts, A. M. 2013. *Nominal Sets*. Number 57 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Schmidt-Schauß, M., Kutsia, T., Levy, J., and Villaret, M. 2017. *Nominal Unification of Higher Order Expressions with Recursive Let.* In *Post-proc. of the 26th Int. Sym. on Logic-Based Program Synthesis and Transformation (LOPSTR 2016)*, volume 10184 of *LNCS*, pp. 328–344. Springer.

Siekmann, J. H. 1979. Matching under commutativity. In *Proc. of the Int. Symposium on Symbolic and Algebraic Manipulation (EUROSAM)*, volume 72 of *LNCS*, pp. 531–545. Springer.

Siekmann, J. H. 1989. Unification Theory. *J. of Sym. Computation*, 7(3-4):207–274.

Urban, C. 2010. *Nominal Unification Revisited.* In *Proc. of the 24th Int. Work. on Unification (UNIF)*, volume 42 of *EPTCS*, pp. 1–11.

Urban, C., Pitts, A. M., and Gabbay, M. J. 2004. *Nominal Unification. Theor. Comput. Sci.*, 323(1-3):473–497.