Formalising Nominal C-Unification Generalised with Protected Variables

Mauricio Ayala-Rincón[1,3]   Washington de Carvalho-Segundo[3]   Maribel Fernández[2]
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

Abstract

This work extends a rule-based specification of nominal C-unification formalised in Coq to include "protected variables" that cannot be instantiated during the unification process. By introducing protected variables, we are able to reuse the C-unification simplification rules to solve nominal C-matching (as well as equality check) problems. From the algorithmic point of view, this extension is sufficient to obtain a generalised C-unification procedure; however, it 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 inference rules and reuse previous formalisations. We also generalise a functional recursive nominal C-unification algorithm specified in PVS with protected variables, effectively adapting this algorithm to the tasks of nominal C-matching and nominal equality check. The PVS formalisation is applied to test the correctness of a Python manual implementation of the algorithm.

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.

When the signature used to build the terms includes commutative[a] operators, these problems are known as nominal C-unification and nominal C-matching, respectively. This paper presents a formalisation in Coq of a set of rules for generalised nominal C-unification and a formalisation in PVS of a recursive generalised nominal C-unification algorithm.

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

---

[a]A set X equipped with a commutative operator $+$ that is closed over X but not necessarily associative defines an algebraic structure $(X, +)$ called commutative magma or commutative groupoid. Commutative magmas have been used to model a variety of problems, including the NAND logic gate and the rock-paper-scissors game.

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 $\exists a.a < 0$ can be written as the nominal term $\exists([a]\mathtt{lt}(a, 0))$, using function symbols $\exists$ and $\mathtt{lt}$ and an abstracted atom a. Notice that $\exists([a]\mathtt{lt}(\mathtt{a}, 0)) \approx^?_\alpha \exists([b]\mathtt{lt}(Y, 0))$ is a nominal unification problem whose solution should be such that a does not occur free in Y.

Nominal unification was proved decidable by Urban et al. (2004), and efficient nominal unification algorithms are available (e.g. 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. Ayala-Rincón et al. (2018a) proposed an inductive nominal C-unification algorithm based on a set of inference rules specified in Coq. 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, if $+$ is commutative, the fixed point equation $(a\, b) \cdot X \approx^?_{\alpha,C} X$ has infinite solutions $X/a + b, X/(a + b) + (a + b), \dots$ (see Ayala-Rincón et al. (2017) for a procedure to generate solutions of fixed point equations). A formalisation in Coq of the inductive inference rules, including correctness and completeness proofs, was later adapted to solve C-matching problems (see Ayala-Rincón et al. (2019)).

This paper revisits and generalises the C-matching specification given by Ayala-Rincón et al. (2019). The generalisation considers an additional parameter $\mathscr{X}$ that is a set of *protected variables* given as part of the input problem. The unification process forbids the instantiation of such protected variables. Therefore, the domain of solutions will be disjoint from this set. If the set of protected variables is empty, the inductive 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 input problem, the algorithm solves nominal C-matching problems. Finally, if the set of protected variables consists of all the variables occurring in the input problem, the inductive algorithm becomes a C-equational checker. Although these conclusions are obvious, from the operational point of view, one cannot reuse the formalisation of the nominal C-unification specification straightforwardly (without protected variables) to verify the derived nominal C-matching specification. In this paper, we show how the formalisation of the generalised nominal C-unification algorithm with protected variables was achieved, and how the properties for the particular case of nominal C-matching were derived.

The inference rules specify an inductive algorithm which is non-deterministic per se: several inference 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.

The formalisations were done in PVS, aiming to enrich a nominal library of unification for this proof assistant. In contrast with Ayala-Rincón et al. (2018a), where termination, soundness and completeness of a set of inductive inference rules were verified, such properties were verified in Ayala-Rincón et al. (2019) for the recursive algorithm itself. The formalisation allowed us to obtain verified executable code from the specification, as well as simpler proofs of termination, soundness and completeness.

It is important to stress here that from the inductive definitions of the inference rules in Coq, it is not possible to extract verified executable code directly. Two additional steps are required: specifying a recursive algorithm and proving it computes the same answers as the inductive specification.

In this paper we also present a functional nominal C-unification algorithm with protected variables, derived from the non-deterministic set of inference rules following the technique used by Ayala-Rincón et al. (2019), and prove its correctness extending the results of Ayala-Rincón et al. (2019). We discuss the interesting aspects of the specification and formalisation of this extension.

Additionally, in this paper, we discuss the advantages and the drawbacks of formalising unification via a set of non-deterministic inference rules and via a recursive algorithm, integrating both approaches in a single article.

To summarise, the contributions of this paper are:

- A rule-based specification of nominal C-unification with a parameter for *protected variables*, extending the nominal C-unification specification proposed by Ayala-Rincón et al. (2018*a*). 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.
- A rule-based inductive 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. Since it is derived as a particular case of generalised nominal C-unification, its termination, soundness and completeness follow directly from the above point.
- A recursive nominal C-unification algorithm parameterised on a set of protected variables, which can be used also for the task of matching and equality check. This algorithm has been formalised in PVS.
- An integration and comparison of two approaches to nominal unification: using a non-deterministic set of rules and using a functional recursive algorithm.
- A description of the methodology used to test an implementation of the recursive algorithm in Python by comparing its results with the results obtained by the algorithm verified in PVS.

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 discusses the functional recursive algorithm for nominal C-unification and the adaptations done to handle a set of protected variables. Section 5 describes how we tested the correctness of the Python manual implementation Finally, Section 6 concludes the paper and discusses future work.

2. Background

2.1 Nominal setting

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 $\mathrm{dom}(\pi) := \{a \in \mathscr{A} \mid \pi \cdot a \neq a\}$. 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 kth function symbol that is commutative and $f_j^{\emptyset}$ the jth 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 $\mathrm{id}.X$ 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:

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

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

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 $t = f^C \langle (a\ b).X, \overline{c} \rangle$ and substitution $\sigma = \{X/[b]b\}$, we obtain that $t\sigma = f^C \langle (a\ b) \cdot [b]b, \overline{c} \rangle = f^C \langle [a]a, \overline{c} \rangle$, where f^C is a commutative function symbol 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 $\mathtt{dom}(\pi) = ds(\pi, id)$.

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);

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

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

Figure 1. Rules for the freshness relation.

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

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

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

Figure 2. Rules for the α-equivalence relation.

Urban et al. (2004)]. Among them we have *freshness preservation*: if $\nabla \vdash a \,\#\, s$ and $\nabla \vdash s \approx_\alpha t$, 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.

2.2 Specifying unification via set of rules and via algorithms

We can specify unification in a proof assistant as a set of non-deterministic inference rules or as a recursive algorithm. In a rule-based specification the unification problem is progressively transformed into a simpler one by the rules. This elegant approach has a higher level of abstraction than the algorithmic way, which can simplify the analysis of some computational properties such as correctness and completeness of solutions (see Ayala-Rincón et al. (2016)). However, the rule-based approach has the drawback that from a specification of these non-deterministic rules we cannot extract executable code directly. Instead, from a set of non-deterministic inductive rules one usually obtains a recursive algorithm by providing a heuristic on how to apply the rules and then extracts executable code. In this case, one can use the formalised computational properties of the non-deterministic rules (soundness, completeness, termination...) to prove the corresponding properties for the algorithm. The extracted executable code can, of course, be used directly, but it can also be used to test the correctness of manual implementations of the algorithm, which may contain optimisations and is usually faster.

Remark 9 (Proof assistants)**.** The Coq and the PVS proof assistants support both approaches to formalise unification, although inductive formalisations via set of rules are more common in Coq (e.g. Urban et al. (2004)) and recursive formalisations are more common in PVS (e.g. Ayala-Rincón et al. (2016)).

3. An inductive rule-based nominal C-unification algorithm with protected variables

3.1 Nominal C-unification problems with protected variables

Ayala-Rincón et al. (2018a) proposed an inductive nominal C-unification algorithm which used a set of transformation rules to deal with equations and another set of rules to deal with freshness constraints and contexts. 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 to the equality and freshness rules. This parameter indicates which variables are forbidden to be instantiated. In the Coq specification, the associated rules for the freshness and the equality relation deal with triples and use $\mathscr{X}$ when necessary. Here, to clarify the presentation, we add it to the freshness and equality rules (Figures 3 and 4) and to some Lemmas.

The quadruple that will be associated with a C-equational problem of the form $\langle \nabla, \mathscr{X}, P \rangle$ is $\langle \nabla, \mathscr{X}, id, P \rangle$, where id is the identity substitution. The calligraphic uppercase letters $\mathscr{P}, \mathscr{Q}, \mathscr{R}$ and $\mathscr{S}$ will denote quadruples.

Definition 10 (Unification problem)**.** A *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.

Remark 11. Consider ∇ and ∇' freshness contexts and σ and σ' substitutions. We will use the following notation:

- $\nabla' \vdash \nabla\sigma$ denotes that $\nabla' \vdash a\#X\sigma$ holds for each $(a\#X) \in \nabla$.
- $\nabla \vdash \sigma \approx \sigma'$ denotes that $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all X in $dom(\sigma) \cup dom(\sigma')$.

Definition 12 (Solution for a quadruple)**.** 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 $\nabla \vdash \delta\lambda \approx \sigma$.

Definition 13 (Solution for a C-unification problem with protected variables)**.** A *solution for a C-unification problem with protected 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 $\mathscr{U}_C(\mathscr{P})$.

Remark 14 (Alternative approach to unification with protected variables)**.** Instead of introducing a set of protected variables, $\mathscr{X}$, one could have introduced constants to replace protected variables. This would require an extension of the nominal grammar to allow "moderated constants" $\pi.C_X$ as well as changes in the definition of permutation action and the calculus of freshness and α-equivalence. An extension of the nominal grammar with special function symbols over which the permutations stay suspended would require analogous modifications. Indeed, if no modification were made then $\pi \cdot C_X$ and $\pi' \cdot C_X$ would both evaluate to C_X and no freshness constraint $a\#C_X$ would be added to the context when we encounter a freshness constraint $a\#?C_X$. However, this would lead to wrong results in some applications. For example, assume we have a rewrite rule $R = f\langle Y, Y \rangle \to 0$ and want to rewrite the nominal term $f\langle X, (a\ b).X \rangle$. For this, we need to

match $f\langle Y, Y \rangle$ and $f\langle X, (a\ b).X \rangle$ which generates freshness constraints $a\#X, b\#X$. If we ignore the freshness constraints and simply rewrite $f\langle X, (a\ b).X \rangle$ to 0, we would obtain $f\langle a, b \rangle \to_R 0$ as a particular case, which is wrong.

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 $\text{var}(P)$ or $\text{var}(\mathscr{P})$, respectively.

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

3.2 Simplification rules

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

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

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

Figure 3. Reduction rules for freshness problems.

These reduction rules are applied with the intention of transforming a nominal unification problem into a nominal unification problem consisting only of *fixed point equations*, i.e., equations of the form $\pi.X \approx_? X$, together with a substitution and a freshness context (see Example 27). These rules do not, however, handle unsolvable freshness and equation constraints, which stay in the unification problem until the end of the algorithm (see Example 42). P is called a *fixed point problem* if it is a set of fixed point equations. We will denote by $P_\approx$ and $P_\#$ the sets of equations and freshness constraints in the set P of a unification problem $\langle\nabla, \mathscr{X}, P\rangle$; and, considering a unification problem $\langle\nabla, \mathscr{X}, P\rangle$, we will denote by $P_{fp_\approx}$ the subset of fixed point equations $\pi.X \approx_? X$ of $P_\approx$ that satisfy $X \notin \mathscr{X}$.

Differently from Ayala-Rincón et al. (2018a), rule ($\approx_?\mathbf{inst}$) checks whether the variable X is a protected variable, before applying the instantiation; and there is the rule ($\approx_?\mathbf{fp}$), which solves fixed point equations with protected variables. To see the reason for this rule, consider the fixed point equation $\pi.X \approx_? X$ with $X \in \mathscr{X}$. Since X is a protected variable, it will never be instantiated by any substitution σ (notice that the mentioned infinite solutions of fixed point equations in the presence of commutative symbols are all obtained by instantiating the variable) and therefore $(\pi.X)\sigma \approx_? X\sigma$ is reduced to $\pi.X \approx_? X$, which can be solved by adding $dom(\pi)\#X$ to our context.

We use the rules described in Figure 5, called *unification steps*, which give a strategy for application of rules specified as presented in Figures 3 and 4. 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 exclusively of fixed point equations to which we cannot apply rule ($\approx_?\mathbf{fp}$). This fact is expressed by the condition $P_\approx = P_{fp_\approx}$ in rule ($\upsilon_\#$).

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

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

$$(\approx_? \mathbf{C})\dfrac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{f_k^C\, s \approx_? f_k^C\, t\}\rangle}{\langle \nabla, \mathscr{X}, \sigma, P \cup \{s \approx_? v\}\rangle}, \left\{ \begin{array}{l} \text{where } s = \langle s_0, s_1\rangle \text{ and } t = \langle t_0, t_1\rangle \\ v = \langle t_i, t_{i+1(mod\ 2)}\rangle, i = 0, 1 \end{array}\right\}$$

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

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

$$(\approx_? \mathbf{fp})\dfrac{\langle \nabla, \mathscr{X}, \sigma, P \uplus \{\pi.X \approx_? X\}\rangle}{\langle \nabla \cup \mathtt{dom}(\pi)\#X, \ \mathscr{X}, \ \sigma, \ P\rangle}, \text{ if } X \in \mathscr{X}$$

Figure 4.   Reduction rules for equational problems.

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

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 theorems below summarise the main properties about the inductive rule-based algorithm proposed. Except when mentioned otherwise, for all the rules with the exception of $(\approx_? \mathbf{fp})$, the proof is similar to the corresponding one in Ayala-Rincón et al. (2018*a*).

Theorem 15 (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. $\mathscr{P} \Rightarrow_\# \mathscr{Q}$).

Proof. By adjusting the proof in Ayala-Rincón et al. (2018*a*) to take into account the set of variables $\mathscr{X}$ and rule $(\approx_? \mathbf{fp})$. □

Theorem 16 (Termination of $\Rightarrow_\approx$, $\Rightarrow_\#$ and $\Rightarrow_\upsilon$). The relations $\Rightarrow_\approx$, $\Rightarrow_\#$ and $\Rightarrow_\upsilon$ are terminating.

Proof. Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P \rangle$ and $\mathscr{Q} = \langle \nabla, \mathscr{X}, \sigma, Q \rangle$ such that $\mathscr{P} \Rightarrow_v \mathscr{Q}$. The proof is by case analysis on the rules of the relation $\Rightarrow_v$ and uses a lexicographic measure over sets of equation and freshness constraints. The measure is given by:

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

For all the rules except $(\approx_? \textbf{fp})$, we simply adjust the proof in Ayala-Rincón et al. (2018a).
For the case of an application of rule $(\approx_? \textbf{fp})$, one observes that:

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

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

Therefore the measure also decreases in this case, which concludes the proof. $\qquad \square$

Theorem 17 (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 proof is essentially the same as the one in Ayala-Rincón et al. (2018a) since the set $\mathscr{X}$ has no effect on the freshness rules. $\qquad \square$

Remark 18. We will call $\mathscr{Q}$ a *leaf* if it is a normal form w.r.t. $\Rightarrow_v$.

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

Lemma 20 (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 $\mathscr{P} \Rightarrow_v \mathscr{Q}$ and $\mathscr{P}$ is valid then $\mathscr{Q}$ is also valid.

Proof. The formalisation is analogous to the corresponding lemma in Ayala-Rincón et al. (2018a). $\qquad \square$

Lemma 21 (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 $\mathscr{P} \Rightarrow_v \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$, then $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$.

Proof. The proof of item (1), for all the rules except $(\approx_? \textbf{fp})$, is similar to the corresponding one in Ayala-Rincón et al. (2018a). For the case of rule $(\approx_? \textbf{fp})$ one needs to conclude the conditions of Definition 12 for the pair $\langle \nabla, \sigma \rangle$ w.r.t. $\mathscr{P}$. Condition (4) is trivially satisfied. The first condition is proved just observing that every constraint $a\#X$ in Δ is also in $\Delta \cup \text{dom}(\pi)\#X$. 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 $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$ using Definition 12, item (2), 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 first proves the statement $X \notin \text{dom}(\sigma)$ using that $X \in \mathscr{X}$ and that, since $\mathscr{P}$ is a valid quadruple, $\mathscr{X} \cap \text{dom}(\sigma) = \emptyset$. From this, $(\pi.X)\sigma$ can be replaced by $\pi.X$ and $X\sigma$ can be replaced by X in the objective $\nabla \vdash (\pi.X)\sigma \approx_{\{\alpha,C\}} X\sigma$, remaining to prove that $\nabla \vdash \pi.X \approx_{\{\alpha,C\}} X$. Then, using the condition (1) of Definition 12 of hypothesis $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$ one has that $\nabla \vdash (\Delta \cup \text{dom}(\pi)\#X)\sigma$. Since $X \notin \text{dom}(\sigma)$, one concludes that $\text{dom}(\pi)\#X \subseteq \nabla$ 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 $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$ with Definition 12, item (3). This finishes the proof of item (1).

Finally, item (2) is by the definition of $\Rightarrow_\upsilon$ and the preservation of solutions by $\Rightarrow_\approx$ (item (1)) and by $\Rightarrow_\#$ (Theorem 17). $\square$

Lemma 22 (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 $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$.

Proof. Once we prove item (1), item (2) will follow from it and Theorem 17. Let's prove item (1). Necessity is proved by case analysis on the derivation rules of $\Rightarrow_\upsilon$ and Lemma 15 is applied to the premise that $\mathscr{P}$ is not a matching leaf to obtain that there exists $\mathscr{Q}'$ such that $\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}'$.

For the case of all rules except $(\approx_? \mathbf{fp})$, the proof is analogous to the corresponding one in Ayala-Rincón et al. (2018a). For the case of rule $(\approx_? \mathbf{fp})$, $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P' \uplus \{\pi.X \approx_? X\} \rangle$ and the quadruple $\mathscr{Q} = \langle \Delta \cup \text{dom}(\pi)\#X, \mathscr{X}, \delta, P' \rangle$ will be a witness. Thus, $\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}$ follows by an application of rule $(\approx_? \mathbf{fp})$. To prove that $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$, one has to show that the conditions of Definition 12 are satisfied, having as hypothesis that $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$. Conditions (2), (3) and (4) are trivially verified. For condition (1), a constraint $a\#X$ is chosen that is in $\Delta \cup \text{dom}(\pi)\#X$ to analyse if it is either in Δ or $\text{dom}(\pi)\#X$. If $a\#X$ is in Δ the proof is trivial, otherwise one first proves the assertion that $X \notin \text{dom}(\sigma)$ from $\mathscr{X} \cap \text{dom}(\sigma) = \emptyset$ (because $\mathscr{P}$ is valid) and the fact that in the application of rule $(\approx_? \mathbf{fp})$ we have $X \in \mathscr{X}$). This allows to replace every $X\sigma$ and every $(\pi.X)\sigma$, respectively, just by X and $\pi.X$, because $X \notin \text{dom}(\sigma)$. Since $\pi.X \approx_? X$ is in $P' \uplus \{\pi.X \approx_? X\}$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$, we have that $\nabla \vdash (\pi.X)\sigma \approx_{\{\alpha,C\}} X$, therefore $\nabla \vdash \pi.X \approx_{\{\alpha,C\}} X$ and then $\text{dom}(\pi)\#X \subseteq \nabla$. On the other hand, having $a \in \text{dom}(\pi)$ 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 $\text{dom}(\pi)\#X \subseteq \nabla$, one concludes.

Sufficiency is formalised as a direct consequence of Lemma 21. $\square$

Theorem 23 (Soundness of $\Rightarrow_\upsilon^*$)**.** Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P \rangle$ be a valid quadruple and $\mathscr{P} \Rightarrow_\upsilon^* \mathscr{Q}$. If $\mathscr{Q}$ is a leaf and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$ then $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$.

Proof. The proof uses Lemmas 20 and 21, and it is done by induction on the number of steps of $\Rightarrow_\upsilon$. If $\mathscr{P} = \mathscr{Q}$ the proof is trivial. In the case where $\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}$, Lemma 21 is applied to conclude. When $\mathscr{P} \Rightarrow_\upsilon \mathscr{R}$ and $\mathscr{R} \Rightarrow_\upsilon^* \mathscr{Q}$, one uses Lemma 21, the induction hypothesis and Lemma 20 to conclude. $\square$

Theorem 24 (Completeness of $\Rightarrow_\upsilon^*$)**.** Let $\mathscr{P} = \langle \Delta, \mathscr{X}, \delta, P \rangle$ be a valid quadruple and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$. Then there exists a leaf $\mathscr{Q}$ such that $\mathscr{P} \Rightarrow_\upsilon^* \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q})$.

Proof. The formalisation follows by well-founded induction on the number of applications of $\Rightarrow_\upsilon$. Also, Lemma 15 is applied in the analysis of the cases where either $\mathscr{P}$ is a leaf or there exists $\mathscr{Q}'$ such that $\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}'$. If $\mathscr{P}$ is a leaf then $\mathscr{P} = \mathscr{Q}$ and the proof is completed. If there exists $\mathscr{Q}'$

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

$$(\upsilon_\approx)^*$$

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

$$(\approx_? C) \qquad\qquad\qquad\qquad (\approx_? C)$$

$$\langle \emptyset, \emptyset, id,\ \left\{ \begin{array}{c} (a\,b).Z \approx_? Z,\ X \approx_? \overline{b},\ Y \approx_? (a\,b).X \\ a\#_?\langle f(Z), [a](\overline{a} * X)\rangle \end{array} \right\}\rangle \qquad \langle \emptyset, \emptyset,\ id,\ \left\{ \begin{array}{c} (a\,b).Z \approx_? Z,\ X \approx_? (a\,b).X,\ Y \approx_? \overline{b}, \\ a\#_?\langle f(Z), [a](\overline{a} * X)\rangle \end{array} \right\}\rangle$$

$$(\upsilon_\approx)^* \text{ and } (\upsilon_\#)^* \qquad\qquad\qquad\qquad (\upsilon_\approx)^* \text{ and } (\upsilon_\#)^*$$

$$\langle \{a\#Z\},\ \emptyset,\ \{X/\overline{b},\ Y/\overline{a}\},\ \{(a\,b).Z \approx_? Z\}\rangle \qquad \langle \{a\#Z\},\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).Z \approx_? Z,\ (a\,b).X \approx_? X\}\rangle$$
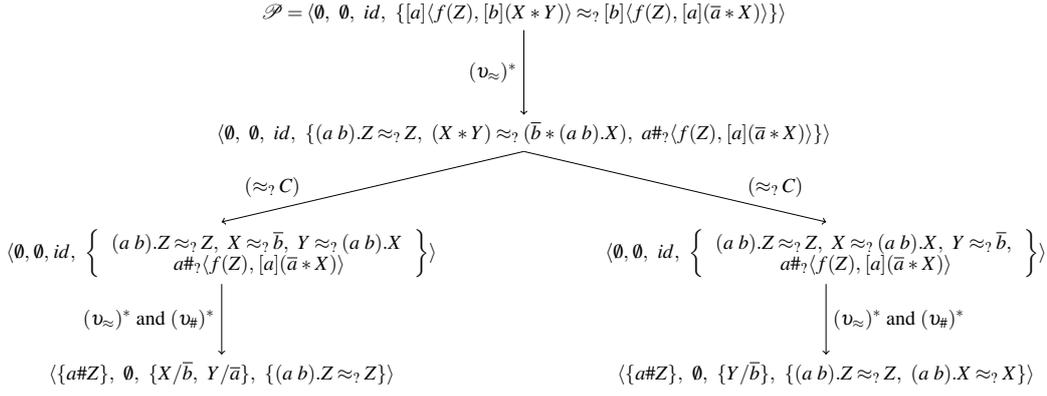
Figure 6. Derivation Tree for Nominal C-unification

such that $\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}'$, one applies Lemma 15 to obtain that $\mathscr{P}$ is not a leaf. Lemma 22 is applied to the premise that $\mathscr{P}$ is not a leaf. From this and the hypothesis $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{P})$ one obtains that there exists $\mathscr{Q}'$ such that $\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}'$.

The induction hypothesis is established as the following statement: $\forall \mathscr{R}$ valid, if $\mathscr{P} \Rightarrow_\upsilon \mathscr{R}$, $\mathscr{X} \cap \text{dom}(\sigma) = \emptyset$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{R})$, then there exists $\mathscr{S}$, such that $\mathscr{R} \Rightarrow_\upsilon^* \mathscr{S}$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{S})$. This is applied to the hypothesis $\mathscr{P} \Rightarrow_\upsilon \mathscr{Q}'$ to conclude that there exists $\mathscr{Q}$, such that $\mathscr{Q}' \Rightarrow_\upsilon^* \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{U}_C(\mathscr{Q}')$. The other premise of induction hypothesis is achieved with the auxiliary result given by Lemma 20. Finally, by case analysis on the statement $\mathscr{Q}' \Rightarrow_\upsilon^* \mathscr{Q}$, one concludes. $\qquad\square$

Definition 25 (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 26 (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 $\text{dom}(\sigma) \cap \mathscr{X} = \emptyset$, then Q is a fixed point problem and it is not possible to apply rule $(\approx_? \mathbf{fp})$ to it.

Proof. Essentially the same as the corresponding one in Ayala-Rincón et al. (2018a), taking into account rule $(\approx_? \mathbf{fp})$. $\qquad\square$

Example 27 (Nominal C-unification with $\mathscr{X} = \emptyset$). This example illustrates the execution of the inductive rule-based nominal C-unification algorithm for the initial problem

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

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 (1) and (2) in the example. The algorithm generates the leaves $\langle \{a\#Z\},\ \emptyset,\ \{X/\overline{b},\ Y/\overline{a}\},\ \{(a\,b).Z \approx_? Z\}\rangle$ and $\langle \{a\#Z\},\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).Z \approx_? Z,\ (a\,b).X \approx_? X\}\rangle$. By Theorem 24, 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 $\langle \{a\#Z\},\ \emptyset,\ \{X/\overline{b},\ Y/\overline{a}\},\ \{(a\,b).Z \approx_? Z\}\rangle$ is a singleton whereas the complete set of solutions of $\langle \{a\#Z\},\ \emptyset,\ \{Y/\overline{b}\},\ \{(a\,b).Z \approx_? Z,\ (a\,b).X \approx_? X\}\rangle$ is infinite. Figure 6 depicts the C-unification derivation tree for $\mathscr{P}$.

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

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

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

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

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

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

(1)

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

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

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

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

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

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

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

(2)

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

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

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

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

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

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

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

3.3 Rule-based nominal C-matching

We now restrict our attention to *nominal C-matching problems*.

Definition 28 (Protected variables and C-matching problems)**.** The set of protected variables for a matching problem $\langle \nabla, P\rangle$ is the set of *right-hand side variables* of the equational constraints in P, denoted by $\texttt{Rvar}(P)$, i.e., $\texttt{Rvar}(P) = \{X \mid s \approx_? t \in P \text{ and } X \in \texttt{var}(t)\}$.

The quadruple associated with the C-matching problem $\langle \nabla, P\rangle$ is given by $\langle \nabla, \texttt{Rvar}(P), id, P\rangle$.

Definition 29 (Solution for a C-matching problem)**.** A *C-matching solution* for a quadruple $\mathscr{P}$ of the form $\langle \Delta, \texttt{Rvar}(P), \delta, P\rangle$ is a pair $\langle \nabla, \sigma\rangle$, where $\texttt{dom}(\sigma) \cap \texttt{Rvar}(P) = \emptyset$, 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 $\langle \Delta, \text{Rvar}(P), id, P \rangle$, its associated C-matching problem. The *solution set* for a matching problem $\mathscr{P}$ is denoted by $\mathscr{M}_C(\mathscr{P})$.

Although standard definitions require that the set of left-hand variables is disjoint from the set of right-hand variables (e.g. Calvès and Fernández (2010)), our definition of nominal matching exclusively demands that substitutions do not instantiate the variables that appear on the right-hand side of the problem. In our Coq specification the inductive rule-based nominal C-matching algorithm consists of applications of *matching step* rules as shown in Figure 7. The set $\mathscr{X}$ of protected variables plays an important role and is fixed as $\text{Rvar}(P)$, i.e., the variables occurring in the right-hand sides of the set of equational constraints P in the input problem. Finally, we observe that if we have in our unification problem the fixed point equation $\pi.X \approx_? X$ then necessarily $X \in \text{Rvar}(P)$. Indeed, the only possibility for a variable X not in the right-hand side of the input problem to appear later on in the right-hand side is if there is an instantiation of a different variable Y, with Y appearing in the right-hand side, to a term that contains X. However, this hypothetical situation could not happen because Y would be protected and therefore, could not be instantiated. With this in mind, we remark that there is now no restriction to apply rule $(\approx_? \mathbf{fp})$, as is shown in Figure 7. We denote derivation with rules of Figure 7 by $\Rightarrow_\mu$.

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

Figure 7. Matching step. There is no restriction to rule $(\approx_? \mathbf{fp})$ anymore.

Remark 30. We will call a quadruple $\mathscr{Q}$ a *matching leaf* if $\mathscr{Q}$ is a normal form w.r.t. $\Rightarrow_\mu$.

3.3.1 Main formalised properties for nominal C-matching

Our Coq specification of matching was obtained by using the unification rules of Ayala-Rincón et al. (2018a), i.e. all rules in Figure 4 except $(\approx_? \mathbf{fp})$, and adding the rule for fixed point equations as shown in Figure 7. Although rule $(\approx_? \mathbf{fp})$ was formalised in Coq only for matching, in this work we opt to present it in the more general context of unification generalised with protected variables (specifically Figure 4), to simplify the presentation. Therefore, although in the Coq formalisation there was more work to adapt the lemmas for nominal C-matching as we needed to consider rule $(\approx_? \mathbf{fp})$, here in this presentation most lemmas follow directly from the corresponding lemmas from nominal C-unification generalised with protected variables.

For a given lemma about matching, we point (when is the case) from which lemma about unification with protected variable it follows.

Lemma 31 ($\mathscr{U}_C$ and $\mathscr{M}_C$ equivalence)**.** Let $\mathscr{P} = \langle \Delta, \text{Rvar}(P), \delta, P \rangle$ 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 $\mathscr{M}_C(\mathscr{P})$. $\qquad\square$

Lemma 32 (Preservation of Rvar by $\Rightarrow_\mu$)**.** Let $\mathscr{P} = \langle \Delta, \text{Rvar}(P), \delta, P \rangle$ and $\mathscr{Q} = \langle \Delta', \text{Rvar}(P), \delta', Q \rangle$ such that $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$. Then $\text{Rvar}(Q) \subseteq \text{Rvar}(P)$.

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

Corollary 33 (Intersection emptiness preservation with right-hand side variables by $\Rightarrow_\mu$). Let $\mathscr{P}$ and $\mathscr{Q}$ be two quadruples, $\langle \Delta, \mathtt{Rvar}(P), \delta, P \rangle$ and $\langle \Delta', \mathtt{Rvar}(P), \delta', Q \rangle$, 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 32.   □

A series of properties related with those given for nominal C-unification with protected variables lead to soundness and completeness of nominal C-matching: Corollary 34 follows from Lemma 20, Lemmas 35 and 37 follow from Lemmas 15 and 21, and Theorems 36, 38, 39, and 40 follow respectively from Theorems 16, Lemma 22, Theorems 23 and 24.

Corollary 34 (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.

Lemma 35 (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.

Theorem 36 (Termination of $\Rightarrow_\mu$). The relation $\Rightarrow_\mu$ is terminating.

Lemma 37 (Preservation of solutions by $\Rightarrow_\mu$). Let $\mathscr{P} = \langle \Delta, \mathtt{Rvar}(P), \delta, P \rangle$ be a valid quadruple and $\mathscr{Q} = \langle \Delta', \mathtt{Rvar}(P), \delta', Q \rangle$. If $\mathtt{Rvar}(P) \cap \mathrm{dom}(\sigma) = \emptyset$, $\mathscr{P} \Rightarrow_\mu \mathscr{Q}$ and $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$, then $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{P})$.

Theorem 38 (Completeness of $\Rightarrow_\mu$). Let $\mathscr{P} = \langle \Delta, \mathtt{Rvar}(P), \delta, P \rangle$ a valid quadruple that is not a matching leaf, if $\mathtt{Rvar}(P) \cap \mathrm{dom}(\sigma) = \emptyset$, 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 $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$.

Theorem 39 (Soundness of $\Rightarrow_\mu^*$). Let $\mathscr{P} = \langle \Delta, \mathtt{Rvar}(P), \delta, P \rangle$ 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 $\mathtt{Rvar}(P) \cap \mathrm{dom}(\sigma) = \emptyset$ then $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{P})$.

Theorem 40 (Completeness of $\Rightarrow_\mu^*$). Let $\mathscr{P} = \langle \Delta, \mathtt{Rvar}(P), \delta, P \rangle$ 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 $\langle \nabla, \sigma \rangle \in \mathscr{M}_C(\mathscr{Q})$.

Theorem 41 (Characterisation of successful matching leaves). Let $\mathscr{Q} = \langle \Delta, \mathtt{Rvar}(P), \delta, Q \rangle$ 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. The formalisation follows from Theorem 26. Since rule ($\approx_? \mathbf{fp}$) can be applied to all fixed point equations, we obtain $Q = \emptyset$.   □

Example 42 (Nominal C-matching). This example illustrates that the simplification rules do not handle unsolvable freshness and equations constraints.

Consider the matching problem

$$\mathscr{P} = \langle \emptyset, \ \{X\}, \ id, \ \{\langle \bar{a}, f\langle (b\ d).X, [d]\bar{d}\rangle\rangle \approx_? \langle \bar{b}, f\langle X, [d]\bar{d}\rangle\rangle\}\rangle.$$

The execution of the algorithm for this matching problem is shown below:

$$\mathscr{P} = \langle \emptyset, \ \{X\}, \ id, \ \{\langle \bar{a}, f\langle (b\ d).X, [d]\bar{d}\rangle\rangle \approx_? \langle \bar{b}, f\langle X, [d]\bar{d}\rangle\rangle\}\rangle$$

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

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

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

$\Rightarrow_{(\approx_? [\mathbf{aa}])} \langle \emptyset, \{X\}, id, \{\overline{a} \approx_? \overline{b}, (b\ d).X \approx_? X, \overline{d} \approx_? \overline{d}\}\rangle$

$\Rightarrow_{(\approx_? \mathbf{refl})} \langle \emptyset, \{X\}, id, \{\overline{a} \approx_? \overline{b}, (b\ d).X \approx_? X\}\rangle$

$\Rightarrow_{(\approx_? \mathbf{fp})} \langle \{b\#X, d\#X\}, \{X\}, id, \{\overline{a} \approx_? \overline{b}\}\rangle$

We end with only one leaf $\langle \{b\#X, d\#X\}, \{X\}, id, \{\overline{a} \approx_? \overline{b}\}\rangle$, which is unsolvable, since no $\langle \nabla, \sigma \rangle$ can satisfy $\nabla \vdash \overline{a}\sigma \approx_{\{\alpha, C\}} \overline{b}$.

3.4 Nominal C-equivalence checking

The generalised rule-based nominal C-unification algorithm can be used as a nominal C-equivalence checker by setting $\mathscr{X}$ to be the set of variables in the input problem. Example 43 illustrates this.

Example 43 (Nominal C-equivalence checking).   This example exhibits the execution of the inductive nominal C-unification algorithm applied to nominal C-equivalence check. It contains two items, (a) and (b):

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

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:

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

and

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

Both are quadruples that have equations without solutions. In the former one, the X cannot be instantiated to solve $X \approx_? \overline{b}$ and neither X nor Y can be instantiated to solve $Y \approx_? (a\ b).X$. 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:

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

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 $\overline{b} \approx_? (a\ b).X$ the 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)

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

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

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

    (2)

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

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

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

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

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

(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)

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

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

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

    (2)

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

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

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

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

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

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

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

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

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

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

The set of inference rules described in Section 3 can be turned into an algorithm by putting the constraints into a list and applying the convenient rule that simplifies the constraint in the head of the list. When no rule can be applied, the algorithm returns `nil`. This strategy was used to obtain a functional algorithm for nominal C-unification in Ayala-Rincón et al. (2019). In this section we extend this algorithm to deal with protected variables 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 44. In contrast with the rule-based algorithm of Section 3, 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: $\text{UNIFY}(\emptyset, \emptyset, id, [(t, s)], \emptyset)$. 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 rule-based algorithm of Section 3.

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.

4.1 Main algorithm and modifications made

The pseudocode for the algorithm is presented in Algorithm 1. Although in the PVS specification all fixed point equations are stored in $FPEqLst$, in the pseudocode here presented we show how fixed point equations $\pi.X \approx X$ with $X \in \mathscr{X}$ can be solved. In relation to the algorithm presented in Ayala-Rincón et al. (2019), there are three changes. First, the addition of the parameter $\mathscr{X}$ for a set of protected variables, which remains constant in the execution of the algorithm. Second there is the check to see if X is in $\mathscr{X}$ or not in lines 6 and 24 to decide whether there will be an instantiation or not. Third, the algorithm solves fixed point equations with protected variables in lines 30−33.

Algorithm 1 - First Part - Functional nominal C-unification

1: **procedure** UNIFY($\mathscr{X}, \Delta, \sigma, PrbLst, FPEqLst$)
2:     **if** nil?($PrbLst$) **then**
3:         **return** list(($\Delta, \sigma, FPEqLst$))
4:     **else**
5:         cons($(t, s), PrbLst'$) = $PrbLst$
6:         **if** (s matches $\pi.X$) and (X not in t) and (X not in $\mathscr{X}$) **then**
7:             $\sigma' = \{X/\pi^{-1} \cdot t\}$
8:             $\sigma'' = \sigma' \circ \sigma$
9:             $(\Delta', bool1)$ = fresh_subs?(σ', Δ)
10:            $\Delta'' = \Delta \cup \Delta'$
11:            $PrbLst'' = $ append($(PrbLst')\sigma', (FPEqLst)\sigma'$)
12:            **if** $bool1$ **then return** UNIFY($\mathscr{X}, \Delta'', \sigma'', PrbLst'', nil$)
13:            **else return** nil
14:            **end if**
15:        **else**
16:            **if** t matches $\bar{a}$ **then**
17:                **if** s matches $\bar{a}$ **then**
18:                    **return** UNIFY($\mathscr{X}, \Delta, \sigma, PrbLst', FPEqLst$)
19:                **else**
20:                    **return** nil
21:                **end if**
22:            **else if** t matches $\pi.X$ **then**
23:                **if** X not in s **then**
24:                    **if** X in $\mathscr{X}$ **then**
25:                        **return** nil
26:                    **else**
27: $\triangleright$ Similar to case above where s is a suspension
28:                    **end if**
29:                **else if** (s matches $\pi'.X$) **then**
30:                    **if** X in $\mathscr{X}$ **then**
31:                        $\Delta' = ds(\pi, \pi')\#X$
32:                        $\Delta'' = \Delta \cup \Delta'$
33:                        **return** UNIFY($\mathscr{X}, \Delta'', \sigma, PrbLst', FPEqLst'$)
34:                    **else**
35:                        $FPEqLst' = FPEqLst \cup \{ \pi.X \approx_? \pi'.X \}$
36:                        **return** UNIFY($\mathscr{X}, \Delta, \sigma, PrbLst', FPEqLst'$)
37:                    **end if**
38:                **else return** nil
39:                **end if**
40:            **else if** t matches $\langle \rangle$ **then**
41:                **if** s matches $\langle \rangle$ **then**
42:                    **return** UNIFY($\mathscr{X}, \Delta, \sigma, PrbLst', FPEqLst$)
43:                **else return** nil
44:                **end if**
45:            **else if** t matches $\langle t_1, t_2 \rangle$ **then**
46:                **if** s matches $\langle s_1, s_2 \rangle$ **then**
47:                    $PrbLst'' = cons((s_1, t_1), cons((s_2, t_2), PrbLst'))$
48:                    **return** UNIFY($\mathscr{X}, \Delta, \sigma, PrbLst'', FPEqLst$)
49:                **else return** nil
50:                **end if**

Algorithm 1 - **Second Part** - Functional nominal C-unification

51:	**else if** t matches $[a]t_1$ **then**
52:	**if** s matches $[a]s_1$ **then**
53:	$PrbLst'' = cons((t_1, s_1), PrbLst')$
54:	**return** UNIFY($\mathcal{X}, \Delta, \sigma, PrbLst'', FPEqLst$)
55:	**else if** s matches $[b]s_1$ **then**
56:	$(\Delta', bool1) = fresh?(a, s_1)$
57:	$\Delta'' = \Delta \cup \Delta'$
58:	$PrbLst'' = cons((t_1, \ (a \ b) \ s_1), PrbLst')$
59:	**if** $bool1$ **then**
60:	**return** UNIFY($\mathcal{X}, \Delta'', \sigma, PrbLst'', FPEqLst$)
61:	**else return** nil
62:	**end if**
63:	**else return** nil
64:	**end if**
65:	**else if** t matches $f \ t_1$ **then** $\triangleright$ f is not commutative
66:	**if** s matches $f \ s_1$ **then**
67:	$PrbLst'' = cons((t_1, s_1), PrbLst')$
68:	**return** UNIFY($\mathcal{X}, \Delta, \sigma, PrbLst'', FPEqLst$)
69:	**else return** nil
70:	**end if**
71:	**else** $\triangleright$ t is of the form $f^C(t_1, t_2)$
72:	**if** s matches $f^C(s_1, s_2)$ **then**
73:	$PrbLst_1 = cons((s_1, t_1), cons((s_2, t_2), PrbLst'))$
74:	$sol_1 = $ UNIFY($\mathcal{X}, \Delta, \sigma, PrbLst_1, FPEqLst$)
75:	$PrbLst_2 = cons((s_1, t_2), cons((s_2, t_1), PrbLst'))$
76:	$sol_2 = $ UNIFY($\mathcal{X}, \Delta, \sigma, PrbLst_2, FPEqLst$)
77:	**return** APPEND(sol_1, sol_2)
78:	**else return** nil
79:	**end if**
80:	**end if**
81:	**end if**
82:	**end if**
83:	**end procedure**

4.2 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) recursively 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) recursively computes and returns the minimal context (Δ' in Algorithm 1) in which a is fresh in t.

The two functions also return a Boolean ($bool1$ in Algorithm 1), to indicate if it was possible to find the mentioned context.

4.3 Interesting points on adapting the algorithm to handle protected variables

After the addition of protected variables, we formalised soundness (Corollary 47) and completeness (Corollary 49) of the algorithm for matching. These corollaries rely, respectively, on Theorems 46 and 48. Since the algorithm has an extra parameter to represent fixed point equations, the definition of valid quadruples is adapted to valid quintuples (Definition 45).

Definition 45 (Valid quintuple)**.** We say $\mathscr{P} = \langle \mathscr{X}, \Delta, \sigma, PrbLst, FPEqLst \rangle$ is a valid quintuple if $\mathrm{im}(\sigma) \cap \mathrm{dom}(\sigma) = \emptyset$ and $\mathrm{dom}(\sigma) \cap (\mathrm{var}(PrbLst) \cup \mathrm{var}(FPEqLst)) = \emptyset$

Theorem 46 (Main theorem for soundness of UNIFY)**.** Suppose $(\Delta_{sol}, \sigma_{sol}, FPEqLst_{sol}) \in$ UNIFY$(\mathscr{X}, \Delta, \sigma, PrbLst, FPEqLst)$, (∇, δ) is a solution to $\langle \mathscr{X}, \Delta_{sol}, \sigma_{sol}, \emptyset, FPEqLst_{sol} \rangle$ and $\langle \mathscr{X}, \Delta, \sigma, PrbLst, FPEqLst \rangle$ is a valid quintuple. Then (∇, δ) is a solution to $\langle \mathscr{X}, \Delta, \sigma, PrbLst, FPEqLst \rangle$.

Proof. The proof is essentially the same as the corresponding theorem for C-unification in Ayala-Rincón et al. (2019). □

Corollary 47 (Soundness of UNIFY for matching)**.** Suppose (∇, δ) is a solution to $\langle \mathrm{var}(s), \Delta_{sol}, \sigma_{sol}, \emptyset, FPEqLst_{sol} \rangle$, and $(\Delta_{sol}, \sigma_{sol}, FPEqLst_{sol}) \in$ UNIFY$(\mathrm{var}(s), \emptyset, id, [(t, s)], \emptyset)$. Then (∇, δ) is a solution to $\langle \mathrm{var}(s), \emptyset, id, [(t, s)], \emptyset \rangle$.

Proof. Notice that $\langle \mathrm{var}(s), \emptyset, id, [(t, s)], \emptyset \rangle$ is a valid quadruple. Then, we apply Theorem 46 and prove the corollary. □

Theorem 48 (Main theorem for completeness of UNIFY)**.** Suppose (∇, δ) is a solution to $\langle \mathscr{X}, \Delta, \sigma, PrbLst, FPEqLst \rangle$ and that $\langle \mathscr{X}, \Delta, \sigma, PrbLst, FPEqLst \rangle$ is a valid quintuple. Then, there exists a computed output $(\Delta_{sol}, \sigma_{sol}, FPEqLst_{sol}) \in$ UNIFY$(\mathscr{X}, \Delta, \sigma, PrbLst, FPEqLst)$ such that the solution (∇, δ) is also a solution to $\langle \mathscr{X}, \Delta_{sol}, \sigma_{sol}, \emptyset, FPEqLst_{sol} \rangle$.

Proof. The proof is essentially the same as the corresponding theorem for C-unification in Ayala-Rincón et al. (2019). □

Corollary 49 (Completeness of UNIFY for matching)**.** Suppose (∇, δ) is a solution to the input quintuple $\langle \mathrm{var}(s), \emptyset, id, [(t, s)], \emptyset \rangle$. Then, there exists $(\Delta_{sol}, \sigma_{sol}, FPEqLst_{sol}) \in$ UNIFY$(\mathrm{var}(s), \emptyset, id, [(t, s)], \emptyset)$ such that (∇, δ) is a solution to $\langle \Delta_{sol}, \sigma_{sol}, \emptyset, FPEqLst_{sol} \rangle$.

Proof. Notice that $\langle \mathrm{var}(s), \emptyset, id, [(t, s)], \emptyset \rangle$ is a valid quintuple. Then, we apply Theorem 48 and prove the corollary. □

An interpretation of Corollary 47 is that if (∇, δ) is a matching solution to one of the outputs of the algorithm UNIFY, then it is a matching solution to the original problem. Similarly, Corollary 49 says that if (∇, δ) is a matching solution to the initial problem then it is a solution to one of the outputs of UNIFY.

Finally, possible pitfalls when adapting a recursive formalisation of C-unification to C-matching are described in Remarks 50 and 51.

Remark 50 (Equations constraints with protected variables)**.** If the algorithm encounters non fixed point equations of the form $\pi.X \approx_? s$, where X in $\mathscr{X}$, it cannot simply return an empty list, since their solubility depends on the form of s. Indeed, if s is a non protected moderated variable, say $\pi'.Y$, the equation $\pi.X \approx_? \pi'.Y$ has solutions of the form $Y/(\pi'^{-1} \oplus \pi).X$.

Remark 51 (Considerations on the parameter $\mathscr{X}$). 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. If one is interested only in C-matching, one might wonder if it is not possible to plug in $\mathtt{Rvar}(PrbLst)$ as the set of protected variables $\mathscr{X}$ directly. However, since the proofs of correctness and completeness are done by induction and from one recursive call of the algorithm to another the set $\mathtt{Rvar}(PrbLst)$ may change, this does not work. 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.

4.4 Examples of the algorithm

Example 52 illustrates the execution of the algorithm for unification, while Examples 53 and 54 illustrate the execution of the algorithm for matching.

Example 52 (Recursive nominal C-unification). This example shows how the algorithm proceeds in order to unify $f^C \langle (a\ b).X, \overline{a} \rangle$ and $f^C \langle \overline{a}, \overline{b} \rangle$. Notice we have $\mathscr{X} = \emptyset$ in all calls to the function UNIFY.

$$\mathtt{UNIFY}(\emptyset, \emptyset, \mathrm{id}, [f^C \langle (a\ b).X, \overline{a} \rangle \approx_? f^C \langle \overline{a}, \overline{b} \rangle], \emptyset)$$

$\mathtt{UNIFY}(\emptyset, \emptyset, \mathrm{id}, [(a\ b).X \approx_? \overline{a},\ \overline{a} \approx_? \overline{b}], \emptyset)$ $\mathtt{UNIFY}(\emptyset, \emptyset, \mathrm{id}, [(a\ b).X \approx_? \overline{b},\ \overline{a} \approx_? \overline{a}], \emptyset)$

$\mathtt{UNIFY}(\emptyset, \emptyset, \{X/\overline{b}\}, [\overline{a} \approx_? \overline{b}], \emptyset)$ $\mathtt{UNIFY}(\emptyset, \emptyset, \{X/\overline{a}\}, [\overline{a} \approx_? \overline{a}], \emptyset)$

nil $\mathtt{UNIFY}(\emptyset, \emptyset, \{X/\overline{a}\}, \mathrm{nil}, \emptyset)$

$(\emptyset, \{X/\overline{a}\}, \emptyset)$
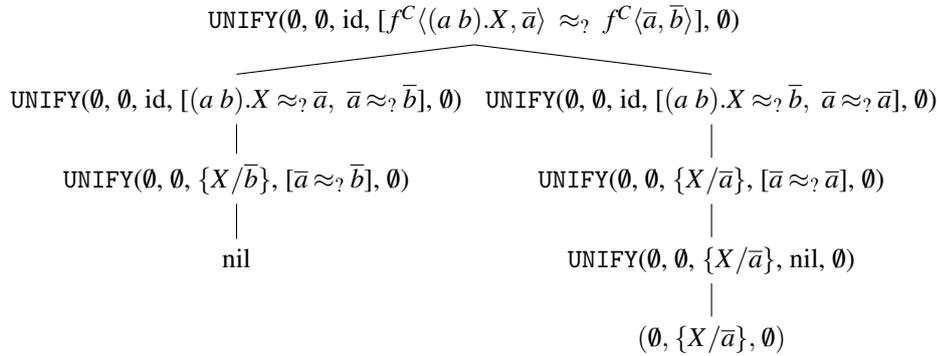
Example 53 (Recursive nominal C-matching). This example is similar to Example 27, but now instead of unification we have matching, as the set of protected variables $\mathscr{X}$ is equal to the right-hand side variables of the initial problem, that is $\{X, Z\}$. The matching problem is:

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

This results in the execution of a nominal C-matching algorithm, with recursive function calls, as shown below:

```
UNIFY({X, Z}, ∅, id, [[a]⟨f(Z), [b](X * Y)⟩ ≈? [b]⟨f(Z), [a](ā * X)⟩], ∅)
  fresh?(a, ⟨f(Z), [a](ā * X)⟩)
    Branch 1:
    fresh?(a, f(Z))
    fresh?(a, Z)
      RETURN({a#Z}, true)

    Branch 2:
    fresh?(a, [a](ā * X))
      RETURN(∅, true)
  RETURN({a#Z}, true)
```

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

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

UNIFY($\{X, Z\}$, $\{a\#Z\}$, id, $[Z \approx_? (a\ b).Z,\ [b](X * Y) \approx_? [b](\overline{b} * (a\ b).X)]$, $\emptyset$)

UNIFY($\{X, Z\}$, $\{a\#Z, b\#Z\}$, id, $[[b](X * Y) \approx_? [b](\overline{b} * (a\ b).X)]$, $\emptyset$)

UNIFY($\{X, Z\}$, $\{a\#Z, b\#Z\}$, id, $[(X * Y) \approx_? (\overline{b} * (a\ b).X)]$, $\emptyset$)

   Branch 1:
   UNIFY($\{X, Z\}$, $\{a\#Z, b\#Z\}$, id, $[(X \approx_? \overline{b},\ Y \approx_? (a\ b).X)]$, $\emptyset$)
     RETURN nil

   Branch 2:
   UNIFY($\{X, Z\}$, $\{a\#Z, b\#Z\}$, id, $[(X \approx_? (a\ b).X,\ Y \approx_? \overline{b}]$, $\emptyset$)

   UNIFY($\{X, Z\}$, $\{a\#Z, b\#Z, a\#X, b\#X\}$, id, $[Y \approx_? \overline{b}]$, $\emptyset$)

   UNIFY($\{X, Z\}$, $\{a\#Z, b\#Z, a\#X, b\#X\}$, $\{Y/\overline{b}\}$, nil, $\emptyset$)
     RETURN ($\{a\#Z, b\#Z, a\#X, b\#X\}$, $\{Y/\overline{b}\}$, $\emptyset$)

RETURN ($\{a\#Z, b\#Z, a\#X, b\#X\}$, $\{Y/\overline{b}\}$, $\emptyset$)

Notice that the algorithm bifurcates in two branches of recursive calls when it encounters an equation constraint $t \approx_? s$ such that t and s are commutative functions headed by the same symbol. The first branch has no solutions, since $X \in \mathscr{X}$ cannot be instantiated to $\overline{b}$, and therefore the algorithm returns nil for this branch. This contrasts with the approach of Section 3, where the inductive algorithm would first simplify $Y \approx_? (a\ b).X$ by instantiating Y and then give the leaf:

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

to which there is no solution. The second branch gives as solution:

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

which is, since the first branch gives no solution, the output returned by the algorithm. The theorems of correctness and completeness guarantee that $\langle \nabla, \sigma \rangle$ is a matching solution to the input problem $\mathscr{P} = \langle \emptyset, \{X, Z\}, id, \{[a]\langle f(Z), [b](X * Y)\rangle \approx_? [b]\langle f(Z), [a](\overline{a} * X)\rangle\}\rangle$ if, and only if, $\langle \nabla, \sigma \rangle$ is a matching solution to the output $\mathscr{Q} = \langle \{a\#Z, b\#Z, a\#X, b\#X\}, \{Y/\overline{b}\}, \emptyset \rangle$.

Example 54 (Unsolvable equation constraints)**.** This example shows how our algorithm handles an unsolvable equation constraint and compares it to the non-deterministic inference rules approach (see Example 42). Therefore, the matching problem is the same as the one in Example 42:

$$\langle \overline{a}, f\langle (b\ d).X, [d]\overline{d}\rangle\rangle \approx_? \langle \overline{b}, f\langle X, [d]\overline{d}\rangle\rangle$$

This results in the execution of the nominal C-matching algorithm, with recursive calls as shown below:

UNIFY($\{X\}$, $\emptyset$, id, $[\langle \overline{a}, f\langle (b\ d).X, [d]\overline{d}\rangle\rangle \approx_? \langle \overline{b}, f\langle X, [d]\overline{d}\rangle\rangle]$, $\emptyset$)

UNIFY($\{X\}$, $\emptyset$, id, $[\overline{a} \approx_? \overline{b}, f\langle (b\ d).X, [d]\overline{d}\rangle \approx_? f\langle X, [d]\overline{d}\rangle]$, $\emptyset$)

```
RETURN nil
```

Notice that as soon as there is an unsolvable equation constraint in the head of *PrbLst* (the list of equations we must still solve) the algorithm returns `nil` communicating that there are no solutions possible for our unification problem. This contrasts with the inference rules approach, where the rules to the unification problem are applied until it is no longer possible (if there is an unsolvable equation constraint in the problem but the rules can be applied to other equational constraints they continue to be applied) and we may have leaves without solution.

4.5 Preserving information regarding protected variables

In our approach, we keep freshness constraints related with protected variables. Such freshness information might be useful in applications, since nominal (C-)matching has direct application in nominal rewriting (which has applications in software engineering, programming languages, etc - see Fernández and Gabbay (2007)). Consider, for instance, the nominal rewriting rule $\oplus\langle Z, Z \rangle \to$ 0 and the terms $\lambda a.a\, X$ and $\lambda b.b\, X$ from the λ-calculus extended with meta-variables. In the nominal framework these terms can be represented as $\mathtt{lam}([a]\mathtt{app}\langle a, id.X \rangle)$ and $\mathtt{lam}([b]\mathtt{app}\langle b, id.X \rangle)$. Then, to check whether the term $\oplus\langle \mathtt{lam}[b]\mathtt{app}\langle b, id.X \rangle, \mathtt{lam}[a]\mathtt{app}\langle a, id.X \rangle\rangle$ reduces with the mentioned nominal rewriting rule one needs to solve the C-matching problem

$$\mathscr{P} = \langle \emptyset, \{X\}, id, \{\oplus\langle Z, Z \rangle \approx_? \oplus\langle \mathtt{lam}[a]\mathtt{app}\langle a, id.X \rangle, \mathtt{lam}[b]\mathtt{app}\langle b, id.X \rangle\rangle\}\rangle.$$

Applying the rule-based nominal C-matching approach of Section 3.3 or the PVS functional C-unification specification, one obtains as output:

$$\langle\{a\#X, b\#X\}, \{X\}, \{Z/\mathtt{lam}[a]\mathtt{app}\langle a, id.X \rangle\}, \emptyset\rangle$$

and

$$\langle\{a\#X, b\#X\}, \{X\}, \{Z/\mathtt{lam}[b]\mathtt{app}\langle b, id.X \rangle\}, \emptyset\rangle$$

Notice that the additional freshness information about protected variables obtained during the generalised C-unification algorithm is necessary. Indeed, by condition (3) of Definition 29 we must have $\nabla \vdash ((a\ b).X)\sigma \approx_{\{\alpha,C\}} X$ and since $\mathtt{dom}(\sigma) \cap \mathtt{Rvar}(P) = \emptyset$ this means that $\nabla \vdash (a\ b).X \approx_{\{\alpha,C\}} X$. According to the rules for the α-equivalence relation, this only holds if $\{a\#X, b\#X\} \subseteq \nabla$.

5. Testing the Python algorithm

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 test if the manual Python algorithm and the formalised algorithm give the same output when run with the same input.

We investigated the literature but could not find a database for unification problems. In Ayala-Rincón et al. (2019) experiments are made for nominal equality-check in the presence of A, C and AC function symbols, while in Calvès and Fernández (2010) experiments are made for syntactic nominal matching and nominal α-equivalence. In Ayala-Rincón et al. (2019), the terms generated are ground and arbitrary choices were made with respect to the size of unification problems, the number of different atoms and the different function symbols. The focus was on the running time of the algorithm. After a term t is randomly generated, the term s of the unification problem

$t \approx_? s$ is generated by swapping arguments of commutative functions and changing the atom being abstracted in an abstraction.

In Calvès and Fernández (2010), experiments were made with syntactic nominal α-equivalence and ground matching problems (i.e. matching problems where there are no variables on the right hand side). The experiments were restricted to solvable problems. The focus was seeing how the running time of the algorithm depends on the size of the unification problem and the type of task (α-equivalence or matching).

In both cases, the terms generated were synthetic and some arbitrary choices were made (although these choices can be manually altered in the code, if one wants). In our tests, some arbitrary choices are also made during the term generation, which we describe now. Our approach covers the approach of Ayala-Rincón et al. (2019) as we swap arguments of commutative functions and change atoms being abstracted in an abstraction. In contrast with Calvès and Fernández (2010) we generate both solvable and unsolvable unification problems.

To compare the Python and the PVS implementation, we generated 2000 unification problems, consisting of terms t and s to be unified and ran the implementations. By printing the Python results in the same way as the PVS implementation prints, it was possible to check whether the implementations match. We generate the term t randomly, with the same probability of generating each component of the grammar of nominal terms, i.e, the probability of generating an atom is the same as the probability of generating a moderated variable and so on. The number of different atoms, variables, function symbols and commutative function symbols was defined arbitrarily to be 10. When generating a permutation for a moderated variable the number of swappings is a random number between 0 and 10.

Finally, we generate the term s as a "copy with modifications" of the term t. These modifications and their corresponding probabilities (chosen arbitrarily) are:

- With a 10% probability we substitute part of the term t by a random moderated variable.

- With a 50% probability, if we encounter a commutative function application in t we change the order of the two arguments.

- With a 50% probability, if we encounter an abstraction $[a]t'$ we change it to a term $[b](a\ b) \cdot t'$.

- With a 10% probability, if we encounter an atom we change it to another atom. Notice that this may result in generating non unifiable terms t and s. This is precisely what we hoped to accomplish, since we also want to test the implementations when the terms are not unifiable.

Both implementations gave the same result for all 2000 unification problems, suggesting that our Python manual implementation is correct. As expected from a manual implementation, the Python code executed faster.

6. Conclusion and Future Work

This paper presents an extension of the inductive rule-based 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. We also tested the Python implementation against the executable code generated by PVS, and our results showed that they give the same output.

A possible path of future work is to 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 can 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 in an actual programming language (in Haskell or OCaml) and be able to compare it with the Python implementation.

Other possible paths of future work include: extending the formalisation to handle permutative equational theories, i.e., equational theories with *n*-ary function symbols with permutative arguments (see Comon (1993)); also, it would be interesting, investigating the formalisation of nominal AC-unification and matching, dealing with restricted cases such as linear AC-matching and, working with unification modulo other equational theories.

Competing Interests: the authors declare none.

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 Unification.* 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 *Post-proc. of the 29th Int. Symp. Logic-based Program Synthesis and Transformation (LOPSTR 2019)*, volume 12042 of *LNCS*, pp. 123–138. Springer.

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. 2010. Matching and Alpha-Equivalence Check for Nominal Terms. *J. of Computer and System Sciences*, 76(5):283–301.

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.

Comon, H. 1993. Complete Axiomatizations of Some Quotient Term Algebras. *Theor. Comput. Sci.*, 118(2):167–191.

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.

Fernández, M. and Gabbay, M. J. 2007. *Nominal Rewriting. Information and Computation*, 205(6):917–965.

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.