

# Reduction Orderings and Completion for Rewrite Systems with Binding

Maribel Fernández<sup>1</sup> and Albert Rubio<sup>2</sup>

<sup>1</sup> King's College London, Strand, London WC2R 2LS, UK  
Maribel.Fernandez@kcl.ac.uk

<sup>2</sup> Technical University of Catalonia, LSI, Jordi Girona 1-3m, 08034 Barcelona, Spain  
rubio@lsi.upc.edu

**Abstract.** We generalise the recursive path ordering (rpo) in order to deal with alpha-equivalence classes of terms, using the *nominal* approach. We then use the nominal rpo to check termination, and to design a completion procedure, for nominal rewriting systems. Nominal rewriting generalises first-order rewriting by providing support for the specification of binding operators — alpha-equivalence is axiomatised, then higher-order reduction schemes can be smoothly represented. Completion of rewriting systems with binding is a notably difficult problem; the completion procedure presented in this paper is the first one that can deal with binders in rewrite rules.

**Key Words:** binders, rewriting, orderings, completion.

## 1 Introduction

Rewriting systems (see, for instance, [18, 2, 37]) have been used to model the dynamics (deduction and evaluation for example) of formal systems described by abstract syntax trees, also called **terms**. In the presence of binding,  $\alpha$ -equivalence, that is, the equivalence relation that equates terms modulo renaming of bound variables, must be taken into account. One alternative is to define binders through functional abstraction, taking  $\alpha$ -equivalence as a primitive, implicit notion, and working with equivalence classes of terms. Combinatory Reduction Systems (**CRS**) [27], Higher-order Rewrite Systems (**HRS**) [30] and Expression Reduction Systems (**ERS**) [26] use the  $\lambda$ -calculus as meta-language; rewriting rules work on  $\alpha$ -equivalence classes of terms. In these systems the  $\lambda$ -calculus can be easily defined with one binder:  $\lambda$ -abstraction. The price to pay is that we can no longer rely on simple notions such as structural induction on terms and syntactic unification.

Alternatively, the nominal approach [23, 34], which we follow in this paper, distinguishes between object-level variables (we write them  $a, b, c$  and call them **atoms**), which can be abstracted but behave similarly to name constants, and meta-level variables or just **variables** (we write them  $X, Y, Z$ ), which are first-order in that there are no binders for them and substitution does not avoid capture of free atoms. In nominal terms (see, for instance, [38] and [19]) variables have arity zero, as in ERSs (but unlike ERSs, substitution of atoms for terms is

not a primitive notion). The  $\alpha$ -equivalence relation is axiomatised in a syntax-directed manner (thus we can reason by structural induction) using a **freshness relation** between atoms and terms, written as  $a\#t$  (i.e., “ $a$  is fresh for  $t$ ”). Nominal rewriting systems (**NRSs**) [19] are rewriting systems on nominal terms;  $\beta$ -,  $\eta$ - and other reduction rules can be easily defined as nominal rewrite rules. For example, the  $\beta$ -reduction rule and the  $\eta$ -expansion rule of the  $\lambda$ -calculus [7] are written as:

$$\begin{array}{ccc} & app(\lambda([a]M), N) & \rightarrow subst([a]M, N) \\ a\#X \vdash & X & \rightarrow \lambda([a]app(X, a)) \end{array}$$

where the substitution in the  $\beta$ -rule is represented by a term-former, also defined by rewrite rules. For instance, we can add the following rules, where we sugar  $subst([a]M, N)$  to  $M\{a \mapsto N\}$ , to propagate substitutions avoiding capture:

$$\begin{array}{ccc} (\sigma_{\text{var}}) & a\{a \mapsto X\} & \rightarrow X \\ (\sigma_{\epsilon}) & a\#Y \vdash Y\{a \mapsto X\} & \rightarrow Y \\ (\sigma_{\text{app}}) & app(X, X')\{a \mapsto Y\} & \rightarrow app(X\{a \mapsto Y\}, X'\{a \mapsto Y\}) \\ (\sigma_{\lambda}) & b\#Y \vdash (\lambda[b]X)\{a \mapsto Y\} & \rightarrow \lambda[b](X\{a \mapsto Y\}) \end{array}$$

We refer to [19] for more examples of nominal rewriting rules.

A step of nominal rewriting involves matching modulo  $\alpha$ -equivalence, which is decidable [38]. For arbitrary NRSs, checking whether there is a rule that can be applied to a given term is an NP-complete problem in general [14]. However, if we only use **closed** rules, nominal matching can be implemented in linear time and space [13]. Closed rules are, roughly speaking, rules which preserve abstracted atoms during reductions (as in the examples above). CRSs, ERSs, and HRSs impose similar conditions on rules, by definition (ERSs impose a condition on matching substitutions, which corresponds to our notion of closed rules). We refer to [21] for an encoding of CRSs using closed nominal rules.

In addition to efficient matching, closed NRSs inherit other good properties of first-order rewriting: for instance, we have a critical pair lemma (see [19]) which can be used to derive confluence of terminating systems. Confluent and terminating NRSs with closed rules have a decidable equational theory [20] (see [22, 15] for definitions and examples of nominal equational theories). However, confluence and termination are both undecidable properties. Sufficient conditions for confluence are given in [19], but so far no techniques are available to prove termination of NRSs. In this paper, we generalise the recursive path ordering (rpo) [17] to deal with nominal terms and  $\alpha$ -equivalence. We show that the nominal recursive path ordering inherits the properties of the rpo, and can be used to check termination of NRSs. We then use this ordering to design a completion procedure à la Knuth and Bendix [28] for closed NRSs.

The principle behind completion is that if a given equational theory is presented by a terminating but not confluent rewrite system, then we can try to transform it into a confluent one by computing its critical pairs and adding rules to join them, preserving termination (but completion may fail, or may not terminate). Completion has been generalised to systems that use higher-order

functions but no binders, i.e. with a first-order syntax [29]. In the case of higher-order rewriting systems, not only we need an ordering that can deal with terms including binders, but also, after computing a critical pair, we need to be able to add the corresponding rules if the pair is not joinable. Adding these rules may not always be possible, as mentioned in [33], due to the syntactic or type restrictions used in higher-order rewriting formalisms. So far, no completion procedures are available for CRSs, ERSs or HRSs. In this paper, we show that a completion procedure can indeed be defined for NRSs when the rules are closed. This result opens the way for the development of tools for automated reasoning in equational theories that include binders.

*Related work.* Algebraic  $\lambda$ -calculi [10, 11, 24, 6, 5] combine the  $\beta$ -reduction rule of the  $\lambda$ -calculus with a set of term rewriting rules, using capture-avoiding substitution in  $\beta$ -reductions and first-order matching and substitution for term rewriting rules. A powerful generalisation of the recursive path ordering to deal with algebraic  $\lambda$ -terms was first presented by Jouannaud and Rubio and called the *higher-order recursive path ordering* (HORPO) (see [25] and [8]).

The higher-order recursive path ordering can also be applied to other higher-order rewriting formalisms, such as CRSs [27], HRSs [30], ERSs [26], HORSs [36], which extend first-order rewriting to include binders using higher-order substitutions and higher-order matching. NRSs also permit binders in left-hand sides, however, nominal rewriting does not use higher-order matching; instead, it relies on nominal matching.

The nominal recursive path ordering (nrpo), in contrast to the HORPO, does not include  $\beta$ -reduction. Hence, in this sense nrpo is less powerful than HORPO. However, thanks to this, nrpo can be more powerful handling the  $\alpha$ -equivalence relation generated by binders, which makes it more powerful for NRSs. On the other hand, nrpo and HORPO differ also in the way both handle the binders. In nrpo we take advantage from the fact that the set of atoms and the set of variables are disjoint.

*Overview of the paper* Section 2 recalls the notions of nominal terms and nominal rewriting, to make the paper self-contained. In Section 3 we define orderings for nominal terms. In Section 4 we define a completion algorithm for NRSs. We conclude the paper in Section 5.

## 2 Preliminaries

A **nominal signature**  $\Sigma$  is a set of **term-formers** (or **function symbols**)  $f, g, \dots$ , each with a fixed arity. In the following, we fix a countably infinite set  $\mathcal{X}$  of **variables**  $X, Y, Z, \dots$ , and a disjoint countably infinite set  $\mathcal{A}$  of **atoms**  $a, b, c, \dots$ . We write  $a \equiv a$  and  $X \equiv X$  to denote syntactic identity.

A **swapping** is a pair of atoms, which we write  $(a\ b)$ . **Permutations**  $\pi$  are represented by lists of swappings, generated by the grammar:  $\pi ::= \text{Id} \mid (a\ b)\pi$ . We usually omit the last  $\text{Id}$  when we write the list of swappings that define

a permutation. We call **Id** the **identity permutation**, and write  $\pi^{-1}$  for the permutation obtained by reversing the list of swappings in  $\pi$ . For example, if  $\pi = (a\ b)(b\ c)$  then  $\pi^{-1} = (b\ c)(a\ b)$ . We denote by  $\pi \circ \pi'$  the permutation containing all the swappings in  $\pi$  followed by those in  $\pi'$ .

**Definition 1.** *Nominal terms*, or just *terms*, are labelled trees generated by the grammar  $s, t ::= a \mid \pi \cdot X \mid [a]s \mid f(s_1, \dots, s_n)$ , and called, respectively, **atoms**, **suspended variables** or simply **variables**, **abstractions** and **function applications** (if  $n = 0$  or  $n = 1$  we may omit the brackets if there is no ambiguity); atoms and variables are leaves. In  $\pi \cdot X$ , we say that  $\pi$  is **suspended** on  $X$  and we abbreviate  $\text{Id} \cdot X$  as  $X$  when there is no ambiguity. We write  $V(t)$  (resp.  $\mathbb{A}(t)$ ) for the set of variables (resp. atoms) occurring in  $t$ ; we will use the same notation for other syntactic objects, such as pairs of terms, contexts, etc. **Ground terms** are terms without variables, that is  $V(t) = \emptyset$ . We denote by  $\text{root}(t)$  the symbol at the root of the term  $t$ .

An abstraction  $[a]t$  is intended to represent  $t$  with  $a$  bound. Accordingly we call occurrences of  $a$  **abstracted** (or bound) if they are under an abstraction, and **unabstracted** (or free) otherwise. We do *not* work modulo  $\alpha$ -equivalence; the  $\alpha$ -equivalence relation  $\approx_\alpha$  will be defined later.

*Example 1.* The ML signature includes term-formers **app** and **let** of arity 2, and **lam** of arity 1. We can write the following terms (the last one is ground): **app**(**lam**( $[a]a$ ),  $X$ ), **lam**( $[a]\text{lam}([b]Y)$ ), **let**( $[a]a$ ,  $a$ ). We will use the following conventions (syntactic sugar): **app**( $s, t$ ) is written  $(s\ t)$ , **lam**( $[a]s$ ) is written  $\lambda[a]s$ , and **let**( $[a]s, t$ ) is written  $\text{let } a = t \text{ in } s$ .

The action of a permutation  $\pi$  on a term  $t$  is defined by induction on the number of swappings in  $\pi$ :  $\text{Id} \cdot t = t$  and  $(a\ b)\pi \cdot t = (a\ b) \cdot (\pi \cdot t)$ , where a swapping acts on terms as follows:

$$\begin{aligned} (a\ b) \cdot a &= b & (a\ b) \cdot b &= a & (a\ b) \cdot c &= c & (c \notin \{a, b\}) \\ (a\ b) \cdot (\pi \cdot X) &= ((a\ b) \circ \pi) \cdot X & (a\ b) \cdot [c]t &= [(a\ b) \cdot c](a\ b) \cdot t \\ (a\ b) \cdot f(t_1, \dots, t_n) &= f((a\ b) \cdot t_1, \dots, (a\ b) \cdot t_n) \end{aligned}$$

**Substitutions** are generated by the grammar  $\sigma ::= \text{Id} \mid [X \mapsto s]\sigma$ . We write substitutions postfix and write  $\circ$  for composition:  $t(\sigma \circ \sigma') \equiv (t\sigma)\sigma'$ .

$$\begin{aligned} a[X \mapsto s] &\equiv a & ([a]t)[X \mapsto s] &\equiv [a](t[X \mapsto s]) \\ f(t_1, \dots, t_n)[X \mapsto s] &\equiv f(t_1[X \mapsto s], \dots, t_n[X \mapsto s]) \\ (\pi \cdot X)[X \mapsto s] &\equiv \pi \cdot s & (\pi \cdot Y)[X \mapsto s] &\equiv \pi \cdot Y \end{aligned}$$

$\sigma$  acts on terms elementwise in the natural way:  $t\text{Id} \equiv t$ ,  $t[X \mapsto s]\sigma \equiv (t[X \mapsto s])\sigma$ .

Substitution and permutation commute:  $\pi \cdot (s\sigma) \equiv (\pi \cdot s)\sigma$ .

**Definition 2.** *Freshness* and  $\alpha$ -*equivalence constraints* have the form  $a\#t$  and  $s \approx_\alpha t$ , respectively, and are specified by rules as follows, where  $a, b$  are any pair of distinct atoms and  $ds(\pi, \pi') = \{a \mid \pi \cdot a \neq \pi' \cdot a\}$  (difference set).

$$\frac{}{a\#b} \text{ (#ab)} \quad \frac{a\#s_1 \cdots a\#s_n}{a\#f(s_1, \dots, s_n)} \text{ (#f)} \quad \frac{}{a\#[a]s} \text{ (#absa)} \quad \frac{a\#s}{a\#[b]s} \text{ (#absb)} \quad \frac{\pi^{-1} \cdot a\#X}{a\#\pi \cdot X} \text{ (#X)}$$

$$\begin{array}{c}
\frac{}{a \approx_{\alpha} a} \text{ } (\approx_{\alpha} \mathbf{a}) \quad \frac{\forall a \in ds(\pi, \pi'), a \# X}{\pi \cdot X \approx_{\alpha} \pi' \cdot X} (\approx_{\alpha} \mathbf{X}) \quad \frac{s_1 \approx_{\alpha} t_1 \cdots s_n \approx_{\alpha} t_n}{f(s_1, \dots, s_n) \approx_{\alpha} f(t_1, \dots, t_n)} (\approx_{\alpha} \mathbf{f}) \\
\frac{s \approx_{\alpha} t}{[a]s \approx_{\alpha} [a]t} (\approx_{\alpha} \mathbf{absa}) \quad \frac{(c a) \cdot s \approx_{\alpha} (c b) \cdot t \quad c \notin \mathbb{A}([a]s, [b]t)}{[a]s \approx_{\alpha} [b]t} (\approx_{\alpha} \mathbf{absb})
\end{array}$$

For example, we can derive  $a\#((a b) \cdot X, (b c) \cdot Y)$  from assumptions  $a\#Y, b\#X$ . Also, we can deduce  $(a b) \cdot X \approx_{\alpha} X$  from assumptions  $a\#X$  and  $b\#X$ , and we also have as expected  $[a]a \approx_{\alpha} [b]b$ . We refer the reader to [19] for more examples.<sup>1</sup>

We write  $\Delta, \nabla, \Gamma$  for sets of constraints of the form  $a\#X$ , called **freshness contexts** or just contexts. We write  $\Delta \vdash Pr$ , for a set of constraints  $Pr$ , and say that  $\Delta$  **entails**  $Pr$ , when proofs of  $P$  exist for all  $P \in Pr$ , using the derivation rules above and elements of the context  $\Delta$  as assumptions. We will write  $\vdash Pr$  instead  $\emptyset \vdash Pr$  when  $\Delta$  is empty.

**Definition 3.** A *nominal rewrite rule*  $R \equiv \nabla \vdash l \rightarrow r$  is a tuple of a context  $\nabla$  and terms  $l$  and  $r$  such that  $V(r, \nabla) \subseteq V(l)$ . A set of rewrite rules is **equivariant** when it is closed under permutations. A **nominal rewrite system** is an equivariant set  $\mathcal{R}$  of nominal rewrite rules. We shall generally equate a set of rewrite rules with its equivariant closure.

*Example 2.* The rules given in the introduction are examples of nominal rewrite rules. We now give rules to compute prenex normal forms in first-order logic using term-formers  $\forall, \exists, \neg, \wedge, \vee$  ( $\wedge$  and  $\vee$  are infix), see also [19]:

$$\begin{array}{ll}
a\#P \vdash P \wedge \forall[a]Q \rightarrow \forall[a](P \wedge Q) & a\#P \vdash (\forall[a]Q) \wedge P \rightarrow \forall[a](Q \wedge P) \\
a\#P \vdash P \vee \forall[a]Q \rightarrow \forall[a](P \vee Q) & a\#P \vdash (\forall[a]Q) \vee P \rightarrow \forall[a](Q \vee P) \\
a\#P \vdash P \wedge \exists[a]Q \rightarrow \exists[a](P \wedge Q) & a\#P \vdash (\exists[a]Q) \wedge P \rightarrow \exists[a](Q \wedge P) \\
a\#P \vdash P \vee \exists[a]Q \rightarrow \exists[a](P \vee Q) & a\#P \vdash (\exists[a]Q) \vee P \rightarrow \exists[a](Q \vee P) \\
\vdash \neg(\exists[a]Q) \rightarrow \forall[a]\neg Q & \vdash \neg(\forall[a]Q) \rightarrow \exists[a]\neg Q
\end{array}$$

We say that a term has a **position** when it mentions a distinguished variable, we usually write it  $Id$ - or just -, precisely once. Below,  $C$  will vary over terms with a position. We write  $C[s]$  for  $C[-\mapsto s]$ , and  $[-]$  when the term  $C$  is precisely its unique variable. For example,  $[a](a, -)$  has a position, but not  $(-, -)$  or  $(a b)-$ .

**Definition 4.** Suppose  $R = \nabla \vdash l \rightarrow r$  is a rewrite rule,  $s$  and  $t$  are terms, and  $\Delta$  is a context, such that  $V(R) \cap V(\Delta, s) = \emptyset$  (we can assume this with no loss of generality). We say that  $s$  **rewrites with  $R$  to  $t$  in  $\Delta$** , and we write  $\Delta \vdash s \xrightarrow{R} t$ , when  $s \equiv C[s']$  and there exists  $\theta$  such that  $\Delta \vdash \nabla \theta$ ,  $\Delta \vdash l\theta \approx_{\alpha} s$  and  $\Delta \vdash C[r\theta] \approx_{\alpha} t$ . Here,  $\theta$  can be found using a nominal matching algorithm [38, 13]. The rewrite relation  $\rightarrow^*$  is the reflexive and transitive closure of this relation. A **normal form** is a term-in-context that does not rewrite.

<sup>1</sup> The version of  $(\approx_{\alpha} \mathbf{absb})$  used here is equivalent to the standard presentations, and it is easier to use in orderings in the next sections.

A nominal rewrite system is **confluent** if, for all  $\Delta, s, t, t'$  such that  $\Delta \vdash s \rightarrow^* t$  and  $\Delta \vdash s \rightarrow^* t'$ , there exists  $u$  such that  $\Delta \vdash t \rightarrow^* u$  and  $\Delta \vdash t' \rightarrow^* u$ . Critical pairs, computed using nominal unification [38, 12], are used to check local confluence, a weaker property, in *closed* NRSs [19] (defined below).

**Definition 5.** *Suppose  $R_i = \nabla_i \vdash l_i \rightarrow r_i$  for  $i = 1, 2$  are copies of two rules in  $\mathcal{R}$  such that  $V(R_1) \cap V(R_2) = \emptyset$  ( $R_1$  and  $R_2$  could be copies of the same rule),  $l_1 \equiv C[l'_1]$  and there is  $(\Gamma, \theta)$  such that  $\Gamma \vdash l'_1 \theta \approx_\alpha l_2 \theta$  and  $\Gamma \vdash \nabla_i \theta$  for  $i = 1, 2$ . Then  $\Gamma \vdash (r_1 \theta, C\theta[r_2 \theta])$  is a **critical pair**. If  $C = [-]$  and  $R_1, R_2$  are copies of the same rule, or if  $l'_1$  is a variable, then the critical pair is **trivial**.*

Given a term in context  $\nabla \vdash t$ , or more generally, a pair  $P \equiv \nabla \vdash (l, r)$  (this could be a rule  $R \equiv \nabla \vdash l \rightarrow r$ ), we shall write  $P^n \equiv \nabla^n \vdash (l^n, r^n)$  to denote a **freshened variant** of  $P \equiv \nabla \vdash (l, r)$  (i.e., a version where the atoms and variables have been replaced by ‘fresh’ ones). We shall always explicitly say what  $P^n$  is freshened for when this is not obvious. For example, a freshened version of  $(a \# X \vdash X \rightarrow X)$  with respect to itself and to the term-in-context  $a' \# X \vdash a'$  is  $(a'' \# X' \vdash X' \rightarrow X')$ , where  $a'' \neq a, a'$  and  $X' \neq X$ . We will write  $A(P') \# V(P)$  to mean that all atoms occurring in  $P'$  are fresh for each of the variables occurring in  $P$ .

**Definition 6.** *Let  $\nabla^n \vdash t^n$  be a freshened version of  $\nabla \vdash t$ . We say that  $\nabla \vdash t$  is **closed** if there exists a substitution  $\sigma$  such that  $\nabla, A(\nabla^n \vdash t^n) \# V(\nabla \vdash t) \vdash \nabla^n \sigma$  and  $\nabla, A(\nabla^n \vdash t^n) \# V(\nabla \vdash t) \vdash t^n \sigma \approx_\alpha t$ . This can be checked using the nominal matching algorithm. This definition applies to nominal rewrite rules, and to pairs  $P \equiv \nabla \vdash (l, r)$  in general, using  $(, )$  as a term-former.*

*Let  $R \equiv \nabla \vdash l \rightarrow r$  be a nominal rewrite rule. We write  $\Delta \vdash s \xrightarrow{R}_c t$  when  $\Delta, A(R^n) \# V(\Delta, s) \vdash s \xrightarrow{R^n} t$  and call this a **closed rewriting step**. Here  $R^n$  is freshened with respect to  $R, \Delta, s$ , and  $t$  (as shown in [19], it does not matter which particular freshened  $R^n$  we choose).*

The intuition is that a closed rule generates the same rewrites with closed rewriting as its (infinitely many) renamings. Although there are interesting systems, such as the axiomatisation of the  $\pi$ -calculus [19], which are not closed, all the systems that arise from functional programming (including the axiomatisation of the  $\lambda$ -calculus) are closed, and all the systems that can be specified in a standard higher-order rewriting formalisms are also closed (see [21]).

We will say that an NRS is **terminating** if all the rewrite sequences are finite. Closed NRSs inherit properties of first-order rewriting systems such as the Critical Pair Lemma: If all non-trivial critical pairs of a closed nominal rewrite system are joinable, then it is locally confluent. If, in addition, it is terminating, then it is confluent.

In the rest of the paper we will always work with closed NRSs, using closed rewriting.

### 3 Reduction orderings for nominal terms

In this section we will define two relations between nominal terms in context, inspired by the recursive path ordering (rpo). The rpo, introduced by Dershowitz [17], is a well-founded ordering on first-order terms based on simple syntactic comparisons, using a precedence on function symbols. It has been widely used to prove termination of first-order rewriting systems.

The first ordering that we will define simply equates all the atoms (by collapsing them into a unique atom  $a$ ) and uses a precedence on the signature  $\Sigma$  of a nominal rewriting system extended with  $a$  and an abstraction operator for  $a$ .

In order to define this ordering, we first define a translation function  $\hat{\cdot}$  such that  $\hat{t}$  coincides with  $t$  except that all the atoms have been replaced by a distinguished atom  $a$ . For example, the translation of  $\text{subst}([b]b, c)$  is  $\text{subst}([a]a, a)$  (we omit the inductive definition of this translation function). Then, to define  $\Delta \vdash s > t$ , we compare  $\hat{s}$  and  $\hat{t}$  using the standard rpo, with a precedence  $>_{\Sigma}$  on  $\Sigma \cup \{a, [a]\cdot\}$ , such that  $f >_{\Sigma} [a]\cdot >_{\Sigma} a$  for all  $f \in \Sigma$ . In other words,  $>$  is simply the rpo using the above precedence on the translated terms. We give a direct definition of  $>$  below, where  $\geq$  is the reflexive closure of  $>$ , and  $>_{mul}$  denotes the multiset extension of  $>$ .

**Definition 7.** *Let  $\Delta$  be a freshness context and  $s, t$  nominal terms over  $\Sigma$ . Let  $>_{\Sigma}$  be a precedence on  $\Sigma$ . We write  $\Delta \vdash s > t$ , if  $\hat{s}$  is greater than  $\hat{t}$  in the recursive path ordering, which is defined by cases as follows:*

1.  $[a]s > t$  if  $s \geq t$
2.  $s > [a]t$  if  $\text{root}(s) \in \Sigma$  and  $s > t$
3.  $[a]s > [a]t$  if  $s > t$
4.  $[a]s > a$
5.  $f(s_1, \dots, s_n) > t$  if  $s_i \geq t$  for some  $i$ ,  $f \in \Sigma$
6.  $f(s_1, \dots, s_n) > g(t_1, \dots, t_m)$  if  $f >_{\Sigma} g$  and  $f(s_1, \dots, s_n) > t_i$  for all  $i$
7.  $f(s_1, \dots, s_n) > a$
8.  $f(s_1, \dots, s_n) > g(t_1, \dots, t_m)$  if  $f =_{\Sigma} g$  and  $\{s_1, \dots, s_n\} >_{mul} \{t_1, \dots, t_m\}$ .

In the definition above, we have not included  $a$  and the abstraction operator in the precedence, but we have given explicitly the cases for comparison with abstraction and atoms (which correspond to  $f > [a]\cdot > a$  in the precedence).

For example, we can use this ordering to orient all the rules in the nominal rewriting system given in Example 2 to compute prenex normal forms of first-order logic formulas. For each rule, we can show  $\Delta \vdash l > r$  if we consider a precedence  $>_{\Sigma}$  such that  $\wedge >_{\Sigma} \forall$ ,  $\wedge >_{\Sigma} \exists$ ,  $\vee >_{\Sigma} \forall$ ,  $\vee >_{\Sigma} \exists$ ,  $\neg >_{\Sigma} \forall$ ,  $\neg >_{\Sigma} \exists$ .

It is easy to see that the ordering defined above satisfies all the required conditions to be used as a reduction ordering:  $>$  is transitive, irreflexive, well-founded, and preserved by context and substitution, because we have  $\Delta \vdash s > t$  if and only if  $\hat{s} >_{rpo} \hat{t}$ . It is also easy to see that  $>$  is preserved by  $\approx_{\alpha}$ , that is, it works uniformly in  $\alpha$ -equivalence classes. This is because all the terms in the  $\alpha$ -equivalence class of  $t$  are mapped to the same term  $\hat{t}$ .

Compared with higher-order versions of the rpo that deal also with  $\beta$ -reduction, this ordering is of course less powerful. But on the other hand, this ordering can compare terms that are not comparable in the higher-order rpo, not even in the parametric version [8], as the following example shows.

*Example 3.* The following example is taken from [8], where it is shown that these terms, or more precisely the corresponding terms in a  $\lambda$ -calculus extended with constants, can be compared under the parametric higher-order rpo (an extension of the higher-order rpo).

To prove  $\vdash f(g([x]f(x, x)), g([x]f(x, x))) > [x]f(x, x)$  we need to show that  $g([a]f(a, a)) > a$ , which is a direct consequence of the fact that  $a$  is smaller than terms rooted by function symbols.

More interestingly, the following terms can be compared using  $>$  but are not comparable under the parametric higher-order rpo:

$$\vdash f(g(X, Y)) > g(X, [a]f(h(Y, a)))$$

To show that this holds, assume  $f >_{\Sigma} g >_{\Sigma} h$ . We show  $\vdash f(g(X, Y)) > X$  and  $\vdash f(g(X, Y)) > [a]f(h(Y, a))$ . The first one is trivial since it is included in the subterm relation. For the second, we show  $\vdash f(g(X, Y)) > f(h(Y, a))$ , which requires  $g(X, Y) > h(Y, a)$ . Since  $g(X, Y) > Y$ , and  $g(X, Y) > a$ , we are done.

It is easy to see that this ordering is not stable under the usual notion of capture-avoiding substitution for atoms (since we are equating all the atoms!). This is not a problem in itself: substitution for atoms is not a primitive operation for nominal terms. Moreover, we can orient with  $>$  the rewrite rules  $\sigma_{\text{var}}, \sigma_{\epsilon}, \sigma_{\text{app}}, \sigma_{\lambda}$  given in the introduction to define the capture-avoiding substitution of the  $\lambda$ -calculus.

However, for certain applications it may be convenient to consider an extended framework including capture-avoiding substitution for atoms as a primitive operation. The ordering above allows us to deal with substitutions for atoms in the  $\beta$ -reduction rule, but it is not suitable if we substitute arbitrary atoms in terms. To address this point, below we define a version of  $>$ , which we will call  $>_{\text{nrpo}}$ , where instead of replacing all atoms by the same one, we replace and equate abstracted atoms. For this we will use a generalisation of  $\approx_{\alpha}$ , denoted by  $\bowtie$ . We start by defining  $\bowtie$ .

**Definition 8.** Let  $\mathbb{C} = \{c_1, c_2, \dots\}$  be an infinite set of distinguished atoms. We define inductively the relation  $\bowtie_{\mathbb{C}}$ , or just  $\bowtie$  if  $\mathbb{C}$  is clear from the context, using the following axioms and rules:

$$\begin{array}{c} \frac{}{a \bowtie a} \text{ (\bowtie a)} \quad \frac{c_i, c_j \in \mathbb{C}}{c_i \bowtie c_j} \text{ (\bowtie c)} \quad \frac{\forall a \in ds(\pi, \pi'), a \# X \text{ or } \pi \cdot a, \pi' \cdot a \in \mathbb{C}}{\pi \cdot X \bowtie \pi' \cdot X} \text{ (\bowtie X)} \\ \\ \frac{s_1 \bowtie t_1 \cdots s_n \bowtie t_n}{f(s_1, \dots, s_n) \bowtie f(t_1, \dots, t_n)} \text{ (\bowtie f)} \quad \frac{s \bowtie t}{[a]s \bowtie [a]t} \text{ (\bowtie absa)} \\ \frac{(c a) \cdot s \bowtie (c b) \cdot t \quad c \in \mathbb{C}, c \notin \mathbb{A}([a]s, [b]t)}{[a]s \bowtie [b]t} \text{ (\bowtie absb)} \end{array}$$

The relation  $\bowtie$  is a generalisation of  $\approx_\alpha$  that equates the atoms in  $\mathbb{C}$ . It has similar properties to  $\approx_\alpha$  regarding freshness constraints and permutations. The proofs of the properties given below are in the appendix.

- Property 1.*
1. If  $\Delta \vdash s \approx_\alpha t$  then  $\Delta \vdash s \bowtie t$ .
  2. If  $\Delta \vdash s \bowtie t$  and  $\Delta \vdash a \# s$  then  $\Delta \vdash a \# t$  or  $a \in \mathbb{C}$ .
  3. If  $\Delta \vdash s \bowtie t$  then  $\Delta \vdash \pi \cdot s \bowtie \pi \cdot t$ , for any  $\pi$  such that  $\forall c \in \mathbb{C}, \pi \cdot c \in \mathbb{C}$  or  $(c \notin \mathbb{A}(s, t) \text{ and } \Delta \vdash c \# s, t)$ .

Using the properties above, we can prove that  $\bowtie$  is an equivalence relation.

*Property 2.* The relation  $\bowtie$  is reflexive, symmetric and transitive.

In order to define  $>_{nrpo}$ , we consider, as above, a precedence  $>_\Sigma$  on the signature  $\Sigma$  of a nominal rewriting system. Also as above, in the definition of the ordering we use its multiset extension. The main differences are that now  $\geq_{nrpo}$  is the union of  $>_{nrpo}$  and  $\approx_\alpha$ , that is, syntactic equality is replaced by the more general  $\approx_\alpha$ , and we define  $>_{nrpo}$  using the auxiliary relation  $\bowtie$ . In the definition of  $>_{nrpo}$  below, we write  $\Delta_{c\#s,t}$  to denote a freshness context such that  $\Delta_{c\#s,t} \vdash c \# s, c \# t$ ; such a context always exists if  $c \notin \mathbb{A}(s, t)$ .

**Definition 9 (nrpo).** *Let  $>_\Sigma$  be a precedence on the signature  $\Sigma$  of a nominal rewriting system,  $\Delta$  a freshness context, and  $s, t$  terms. Let  $\mathbb{C} = \{c_1, c_2, \dots\}$  be a set of atoms such that  $\mathbb{C} \cap \mathbb{A}(\Delta, s, t) = \emptyset^2$ . We define  $\Delta \vdash s >_{nrpo} t$ , or just  $\Delta \vdash s > t$ , by cases below, where  $\geq$  is  $> \cup \bowtie$ .*

1.  $\Delta \vdash [a]s > t$  if  $\Delta \cup \Delta_{c\#s,t} \vdash (a \ c) \cdot s \geq t$ , for an arbitrary  $c \in \mathbb{C} - \mathbb{A}(\Delta, [a]s, t)$ .
2.  $\Delta \vdash s > [a]t$  if  $\text{root}(s) \in \Sigma$  and  $\Delta \cup \Delta_{c\#s,t} \vdash s > (a \ c) \cdot t$ , for an arbitrary  $c \in \mathbb{C} - \mathbb{A}(\Delta, s, [a]t)$ .
3.  $\Delta \vdash [a]s > [b]t$  if  $\Delta \cup \Delta_{c\#s,t} \vdash (a \ c) \cdot s > (b \ c) \cdot t$ , for an arbitrary  $c \in \mathbb{C} - \mathbb{A}(\Delta, [a]s, [b]t)$ .
4.  $\Delta \vdash [a]s > [a]t$  if  $\Delta \vdash s > t$ .
5.  $\Delta \vdash [a]s > c$  if  $c \in \mathbb{C}$ .
6.  $\Delta \vdash f(s_1, \dots, s_n) > t$  if  $\Delta \vdash s_i \geq t$  for some  $i$ .
7.  $\Delta \vdash f(s_1, \dots, s_n) > g(t_1, \dots, t_m)$  if  $f >_\Sigma g$  and  $\Delta \vdash f(s_1, \dots, s_n) > t_i$  for all  $i$ .
8.  $\Delta \vdash f(s_1, \dots, s_n) > c$  if  $c \in \mathbb{C}$ .
9.  $\Delta \vdash f(s_1, \dots, s_n) > g(t_1, \dots, t_m)$  if  $f =_\Sigma g$  and  $\Delta \vdash \{s_1, \dots, s_n\} >_{mul} \{t_1, \dots, t_m\}$ .

Before showing that  $>_{nrpo}$  is indeed an ordering with the desired properties, we give an example of application, where we use  $>_{nrpo}$  to orient the rules in Example 2.

*Example 4.* For all the rules  $\Delta \vdash l \rightarrow r$  in Example 2,  $\Delta \vdash l >_{nrpo} r$  if we consider a precedence  $>_\Sigma$  such that  $\wedge >_\Sigma \forall, \wedge >_\Sigma \exists, \vee >_\Sigma \forall, \vee >_\Sigma \exists, \neg >_\Sigma \forall, \neg >_\Sigma \exists$ . We show the derivation for the first rule only:

<sup>2</sup> Any such set  $\mathbb{C}$  can be used due to the equivariance of the nominal framework.

Since  $\wedge >_{\Sigma} \forall$ , it is sufficient to show  $a\#P \vdash P \wedge \forall[a]Q > [a](P \wedge Q)$  and since we now have an abstraction in the right-hand side, it is sufficient to show

$$a\#P, c\#P, c\#Q \vdash P \wedge \forall[a]Q > (a\ c)\cdot P \wedge (a\ c)\cdot Q.$$

We now use case (9), and compare their subterms. Since  $a\#P, c\#P, c\#Q \vdash P \approx_{\alpha} (a\ c)\cdot P$ , it boils down to proving  $a\#P, c\#P, c\#Q \vdash \forall[a]Q > (a\ c)\cdot Q$ , for which it is sufficient to show  $a\#P, c\#P, c\#Q \vdash [a]Q \geq (a\ c)\cdot Q$ . This is a consequence of  $a\#P, c\#P, c\#Q, c'\#Q \vdash (a\ c')\cdot Q \geq (a\ c)\cdot Q$ , which holds since  $ds((a\ c'), (a\ c)) = \{a, c, c'\}$  and we can use  $(\bowtie\mathbf{X})$  to derive

$$a\#P, c\#P, c\#Q, c'\#Q \vdash (a\ c')\cdot Q \bowtie (a\ c)\cdot Q$$

We can also orient the substitution rules for the  $\lambda$ -calculus (see the Introduction) using a precedence where  $subst >_{\Sigma} app$  and  $subst >_{\Sigma} \lambda$ . The  $\beta$  rule can be oriented with a precedence such that  $app >_{\Sigma} subst$  (as expected, we cannot orient both  $\beta$  and the substitution rules). We give more examples below.

The nominal recursive path ordering  $>_{nrpo}$  is a generalisation of the recursive path ordering to take into account atoms, abstractions, and  $\approx_{\alpha}$ . We use permutations in Definition 9 to ensure that substitutions (of terms for variables) are dealt with correctly, and we use the atoms in  $\mathbb{C}$  to make sure that substitutions of terms for atoms can also be performed, although as we mentioned before this is not a primitive operation in our framework. Freshness contexts are needed to deal with  $\approx_{\alpha}$ . We show below that  $>_{nrpo}$  satisfies the properties of the rpo.

*Property 3 (Reduction Ordering).* The nrpo has the following properties:

1. It is a decidable relation.
2. If  $\Delta \vdash s >_{nrpo} t$  then:
  - (a) For all  $C[-]$ ,  $\Delta \vdash C[s] >_{nrpo} C[t]$ .
  - (b) For all  $\pi$ ,  $\Delta \vdash \pi\cdot s >_{nrpo} \pi\cdot t$ .
  - (c) For all  $\Gamma$  such that  $\Gamma \vdash \Delta\sigma$ ,  $\Gamma \vdash s\sigma >_{nrpo} t\sigma$ .
3. It is an irreflexive and transitive relation.
4. It is well founded when the precedence is well-founded: there are no infinite descending chains  $\Delta \vdash s_1 >_{nrpo} s_2 >_{nrpo} \dots$

*Proof.* 1. Decidability follows from the decidability of nominal constraints [38].

2. The first part follows directly by induction on  $C[-]$ .

We now prove that the ordering is preserved by permutation and substitution. By assumption,  $\mathbb{C} \cap \mathbb{A}(\Delta, s, t) = \emptyset$  and  $\Delta \vdash s > t$ . Without loss of generality we can assume that  $\mathbb{C} \cap \mathbb{A}(\pi, \sigma) = \emptyset$  (this implies that  $\pi\cdot c = c$ ,  $\forall c \in \mathbb{C}$ ). We proceed by induction on the definition of  $>$ . The only interesting cases are the ones that deal with abstractions. We show the first case for both properties:

Assume  $\Delta \vdash [a]s > t$  because  $\Delta \cup \Delta_{c\#s,t} \vdash (a\ c)\cdot s \geq t$  where  $c \in \mathbb{C}$  is a new atom. Also  $\Delta \cup \Delta_{c\#s,t} \vdash c\#s, t$  by definition of  $\Delta_{c\#s,t}$ . We need to prove that  $\Delta \vdash \pi\cdot[a]s > \pi\cdot t$ . Since  $\pi\cdot[a]s = [\pi\cdot a]\pi\cdot s$  and  $\pi\cdot((a\ c)\cdot s) = (\pi\cdot a\ c)\cdot(\pi\cdot s)$  by

assumption, the result follows by induction. Note that in case  $\Delta \cup \Delta_{c\#s,t} \vdash (a\ c)\cdot s \bowtie t$ , we obtain  $\Delta \cup \Delta_{c\#s,t} \vdash \pi\cdot((a\ c)\cdot s) \bowtie \pi\cdot t$  by Property 1.

We also need to prove that  $\Gamma \vdash ([a]s)\sigma > t\sigma$ . Without loss of generality we can assume that  $c$  is fresh also for  $([a]s)\sigma, t\sigma$ . Since  $((a\ c)\cdot s)\sigma = (a\ c)\cdot(s\sigma)$ , the result follows by induction.

3. By induction on the definition of  $>_{nrpo}$ . The proof is similar to the one given by Dershowitz for the rpo in [17]. In the proof of transitivity the interesting cases are the ones that deal with abstractions, for which we use the fact that  $>$  is compatible with  $\bowtie$  (see Lemma 1 below).
4. To prove that the ordering is well-founded, we show that it is included in an ordering that contains the subterm relation, and we can then use Kruskal's Tree Theorem, as in the case of the rpo.

The following lemma shows that  $>$  is uniform on the equivalence classes generated by  $\bowtie$ .

**Lemma 1 ( $>$  is compatible with  $\bowtie$ ).** *Assume  $\Delta \vdash s \bowtie s'$  and  $\Delta \vdash t \bowtie t'$ . If  $\Delta \vdash s > t$  then also  $\Delta \vdash s' > t$  and  $\Delta \vdash s > t'$ .*

*Proof.* In both cases, we proceed by induction on the size of  $s, s', t, t'$ . We distinguish cases according to the definition of  $\Delta \vdash s > t$  (without loss of generality we assume that the set  $\mathbb{C}$  does not include atoms used in  $s', t'$ ).

We show the first case for abstraction (the other cases are similar, or follow directly by induction).

Assume  $\Delta \vdash [a]s > t$  because  $\Delta \cup \Delta_{c\#s,t} \vdash (a\ c)\cdot s \geq t$ , for some fresh  $c \in \mathbb{C}$ . Also,  $\Delta \cup \Delta_{c\#s,t} \vdash c\#s, t$ . We can assume without loss of generality that  $c$  is also fresh for  $s'$  and  $t'$  in  $\Delta \cup \Delta_{c\#s,t}$ .

Since  $\Delta \vdash [a]s \bowtie s'$ , there are two cases:

- $s' \equiv [a]s''$  and  $\Delta \vdash s \bowtie s''$ , in which case  $\Delta \cup \Delta_{c\#s,t} \vdash (a\ c)\cdot s \bowtie (a\ c)\cdot s''$  by Property 1 and therefore by induction (or transitivity of  $\bowtie$  if  $\Delta \vdash (a\ c)\cdot s \bowtie t$ ) we obtain  $\Delta \cup \Delta_{c\#s,t} \vdash (a\ c)\cdot s'' \geq t$ , which implies  $\Delta \vdash s' > t$ .
- $s' \equiv [b]s''$  and  $\Delta \vdash (c\ a)\cdot s \bowtie (c\ b)\cdot s''$  (without loss of generality we can assume the same  $c$  is used here). Then  $\Delta \vdash (c\ b)\cdot s'' \geq t$  by induction (or transitivity of  $\bowtie$ ). Again we conclude that  $\Delta \vdash s' > t$ .

Also, since  $\Delta \vdash t \bowtie t'$ , by induction  $\Delta \cup \Delta_{c\#s,t} \vdash (a\ c)\cdot s \geq t'$ , and this implies  $\Delta \vdash [a]s > t'$ .

As a corollary, we deduce that the nominal recursive path ordering  $>_{nrpo}$  is also  $\alpha$ -compatible, in the following sense:

**Corollary 1 ( $\alpha$ -compatibility).** *Assume  $\Delta \vdash s \approx_\alpha s'$  and  $\Delta \vdash t \approx_\alpha t'$ . If  $\Delta \vdash s >_{nrpo} t$  then also  $\Delta \vdash s' >_{nrpo} t$  and  $\Delta \vdash s >_{nrpo} t'$ .*

*Proof.* Consequence of Lemma 1 and Property 1 ( $\approx_\alpha$  is included in  $\bowtie$ ).

More interestingly,  $>_{nrpo}$  is also stable by the standard capture-avoiding notion of substitution for atoms. Substitutions for atoms cannot be applied on variables, so we will prove this property for ground terms only. In the statement of the property below we will use  $\tau$  to denote a substitution for atoms (not to be confused with the substitution on variables, which is primitive in the nominal framework and does not avoid capture)

*Property 4.* Let  $s, t$  be ground terms. If  $\vdash s >_{nrpo} t$  then also  $\vdash s\tau >_{nrpo} t\tau$ , where  $\tau$  is a capture-avoiding substitution for atoms.

*Proof.* First we show that  $\bowtie$  is preserved by atom substitution. We do this by induction on the definition of  $\bowtie$ , where without loss of generality we assume that  $\mathbb{C} \cap (\text{dom}(\tau) \cup \text{Im}(\tau)) = \emptyset$ , that is,  $\mathbb{C}$  does not contain atoms that will be affected by  $\tau$  or introduced by  $\tau$ . We distinguish cases according to the last rule used to derive  $\vdash s \bowtie t$ . The cases for  $(\bowtie\mathbf{a})$  and  $(\bowtie\mathbf{c})$  are trivial, and the cases for  $(\bowtie\mathbf{f})$  and  $(\bowtie\mathbf{absa})$  follow directly by induction. The interesting case is  $(\bowtie\mathbf{absb})$ . Here, we have  $\vdash (c\ a) \cdot s' \bowtie (c\ b) \cdot t'$ , where  $c$  is fresh and  $s \equiv [a]s', t \equiv [b]t'$ . By induction,  $\vdash ((c\ a) \cdot s')\tau \bowtie ((c\ b) \cdot t')\tau$ . Moreover, since  $c$  is fresh, by definition of capture-avoiding substitution  $([a]s')\tau \approx_\alpha ([c](c\ a) \cdot s')\tau = [c]((c\ a) \cdot s')\tau$  (and similarly for  $[b]t'$ ). Since  $\bowtie$  is preserved by context and includes  $\approx_\alpha$  (by Property 1), and is also transitive (by Property 2), we deduce  $\vdash s\tau \bowtie t\tau$ .

We prove now that  $>_{nrpo}$  is preserved by atom substitution. Assume  $\vdash s > t$  using a set of atoms  $\mathbb{C}$  (without loss of generality we assume that  $\mathbb{C}$  does not contain atoms occurring in  $\tau$ ). We proceed by induction on  $>$ . The only interesting cases are the ones for abstraction. We show the first one:

Suppose  $\vdash s > t$  because  $s \equiv [a]s'$  and  $\vdash (a\ c) \cdot s' \geq t$ . Then, by induction and preservation of  $\bowtie$  by substitution,  $\vdash ((a\ c) \cdot s')\tau \geq t\tau$ . Now, we observe again that  $([a]s')\tau \approx_\alpha ([c](c\ a) \cdot s')\tau = [c]((c\ a) \cdot s')\tau$ , since  $c$  is a fresh atom. We conclude that  $[c]((c\ a) \cdot s')\tau > t\tau$ , and since  $>$  is  $\approx_\alpha$ -compatible (Property 1), we are done.

*Example 5.* We can use  $>_{nrpo}$  to compare the terms  $f(g([x]f(x, x)), g([x]f(x, x)))$  and  $[x]f(x, x)$ , as in Example 3. Indeed, we can show

$$\vdash f(g([x]f(x, x)), g([x]f(x, x))) > [x]f(x, x)$$

as follows: Using case 2 in Definition 9, we need to prove

$$\vdash f(g([x]f(x, x)), g([x]f(x, x))) > f(c, c)$$

for some fresh  $c \in \mathbb{C}$ , which requires  $g([x]f(x, x)) > c$ , and this holds by definition (all terms rooted by a symbol in  $\Sigma$  are greater than  $c$ ).

Unlike  $>$  in Example 3,  $>_{nrpo}$  cannot compare  $f(g(X, Y))$  and  $g(X, [a]f(h(Y, a)))$  in the empty context. This is not surprising, because the relation would not be stable by substitution for variables and atoms if it included these terms. The reason is that, in the empty context, we can instantiate  $Y$  with a term which contains  $a$ . Then the term in the left would have  $a$  free, whereas the one on the right has  $a$  bound.

However, it is possible to compare these terms in a context where  $a$  is fresh for  $Y$ :  $a\#Y \vdash f(g(X, Y)) >_{nrpo} g(X, [a]f(h(Y, a)))$ .

To show that this holds, assume  $f >_{\Sigma} g >_{\Sigma} h$ . Then we show  $a\#Y \vdash f(g(X, Y)) > X$  and  $a\#Y \vdash f(g(X, Y)) > [a]f(h(Y, a))$ . The first one is trivial. For the second, we show  $a\#Y, c\#Y, c\#X \vdash f(g(X, Y)) > f(h((a\ c)\cdot Y, c))$ , which requires  $a\#Y, c\#Y, c\#X \vdash g(X, Y) > (a\ c)\cdot Y$  and  $a\#Y, c\#Y, c\#X \vdash g(X, Y) > c$ . The first holds because  $a\#Y, c\#Y, c\#X \vdash Y \approx_{\alpha} (a\ c)\cdot Y$ , whereas the second holds directly by Definition 9, case 8.

The orderings defined in this section can be used to prove termination of NRSs, as the following theorem states.

**Theorem 1.** *If for all  $\nabla \vdash l \rightarrow r$  in a nominal rewrite system  $\mathcal{R}$  we have  $\nabla \vdash l > r$ , where  $>$  is one of the orderings defined in this section, then  $\mathcal{R}$  is terminating.*

*Proof.* Since the orderings are preserved by  $\approx_{\alpha}$ , substitution and context (see Property 3 and Lemma 1), all rewrite sequences are strictly decreasing. Moreover, since the orderings are well-founded (see Property 3), all rewrite sequences are finite.

Note that the theorem above requires that all the rules be oriented in order to derive the termination of the system, and a nominal rewrite system may have a potentially infinite number of variants of each rule (NRSs are closed under equivariance). However, since the ordering is itself equivariant, it is sufficient to orient one variant of each rule.

## 4 Completion for nominal rewriting systems

Let  $E$  be a nominal equational theory, that is, a set of axioms of the form  $\Delta_i \vdash s_i = t_i$ . If the axioms in  $E$  can be oriented to form a confluent and terminating closed rewrite system, then closed nominal rewriting can be used to efficiently decide equality modulo  $E$  and  $\approx_{\alpha}$  (see [20]). However, finding a set of rewrite rules equivalent to  $E$  and with the right properties is not an easy task.

To prove termination of the oriented equations, we can use the extensions of the rpo defined in Section 3. Then, if the system is terminating, to prove confluence it is sufficient to prove local confluence [31]. If the system is not locally confluent, we can try to complete it. Completion procedures were first described by Knuth and Bendix [28] for first-order rewriting. Completion of higher-order rewriting systems is based on the same principle: a terminating, non-confluent system can be transformed into a confluent one by adding rules to ensure that all the critical pairs are joinable, while preserving termination. Obviously, completion may fail (for instance, we might not be able to orient a critical pair) or may not terminate (generating new critical pairs for the added rules). In the case of higher-order rewriting systems, the difficulty lies not only in the definition of a suitable ordering, but also in that after computing an

orientable critical pair, we might not be able to add the corresponding rule due to syntactic or type restrictions. So far, no completion procedures are available for CRSs, ERSs or HRSs.

In the case of NRSs, a critical pair generated from two closed rewrite rules is also closed (we prove this property below). Then, given a closed rewrite system, we can complete it by computing all critical pairs and checking their joinability (i.e., that both terms in the pair have the same normal form, which exists if the term rewriting system is terminating). If they are not joinable, then the pair will be added as a new rule, provided it can be oriented.

In order to prove that all the rules generated by the completion procedure are closed, we need some lemmas.

**Lemma 2.** *Assume  $\Delta \vdash t$  is closed. If  $\Delta \vdash t \approx_\alpha t'$  then  $\Delta \vdash t'$  is closed.*

*Proof.* Consequence of the fact that Definition 6 takes into account  $\approx_\alpha$ .

**Lemma 3.** *1. If  $\Delta \vdash (s, t)$  is a critical pair from a closed nominal rewriting system, then  $\Delta \vdash (s, t)$  is closed.*

*2. If  $\Delta \vdash s$  is closed and  $\Delta \vdash s \xrightarrow{R}_c t$  with a closed rule  $R$  then  $\Delta \vdash (s, t)$  is closed.*

*3. If  $\Delta \vdash (s, t)$  is closed and  $\Delta \vdash t \xrightarrow{R}_c u$  using a closed rule, then  $\Delta \vdash (s, u)$  is closed.*

*Proof.* (Sketch) We use a structural characterisation of closedness [16]. A term in context  $\Delta \vdash t$  is closed if:

1. Every atom  $a$  in  $t$  is below an abstraction  $[a]\cdot$  in the syntax tree.
2. For any atom  $a$ , if  $a$  is bound in an occurrence of  $X$  in  $t$  (i.e., if there is an abstraction  $[\pi \cdot a]$  above  $\pi \cdot X$  in the syntax tree), then either it is bound in all occurrences of  $X$  in  $t$  or  $a \# X \in \Delta$ .
3. If  $\pi_1 \cdot X, \pi_2 \cdot X$  occur in  $t$ , then for all  $a \in ds(\pi_1, \pi_2)$ , if  $a$  is not bound in one of the occurrences then  $a \# X \in \Delta$ .

The first property can be proved directly by observing that the structural conditions above hold for  $\Delta \vdash (s, t)$ , using the fact that critical pairs are computed using versions of the rules with disjoint variables, and the substitution used to generate the critical pair is a principal unifier.

To prove the second property we use Lemma 2, and rely on the fact that closed rules do not contain occurrences of free atoms, by definition of rewrite rule any variable occurring in the right-hand side of a rule must also occur in the left-hand side, and closed rewriting is uniform [19], that is, any atom that is fresh in  $s$  remains fresh in  $t$ .

The third property then follows from the fact that  $\Delta \vdash (t, u)$  is also closed, using the second property.

Below we specify a completion procedure for NRSs as a set of transformation rules on pairs  $(E, \mathcal{R})$  consisting of a nominal equational theory and a set of

nominal rewriting rules, in the style of Bachmair et al. [3] (see also [4]) to produce an inter-reduced system.

Input:

$(E, \emptyset)$  and a (well-founded) reduction ordering  $>$ , for instance the nrpo.

Transformation rules:

$$\begin{aligned}
(E, \mathcal{R}) &\Rightarrow (E \cup \Delta \vdash s = t, \mathcal{R}) \text{ if } \Delta \vdash (s, t) \text{ is a critical pair of } \mathcal{R} \\
(E \cup \Delta \vdash s = t, \mathcal{R}) &\Rightarrow (E, \mathcal{R} \cup \Delta \vdash s \rightarrow t) \text{ if } \Delta \vdash s > t \\
(E \cup \Delta \vdash s = t, \mathcal{R}) &\Rightarrow (E, \mathcal{R} \cup \Delta \vdash t \rightarrow s) \text{ if } \Delta \vdash t > s \\
(E \cup \Delta \vdash s = s, \mathcal{R}) &\Rightarrow (E, \mathcal{R}) \\
(E \cup \Delta \vdash s = t, \mathcal{R}) &\Rightarrow (E \cup \Delta \vdash s' = t, \mathcal{R}) \text{ if } \Delta \vdash s \xrightarrow{\mathcal{R}} s' \\
(E \cup \Delta \vdash s = t, \mathcal{R}) &\Rightarrow (E \cup \Delta \vdash s = t', \mathcal{R}) \text{ if } \Delta \vdash t \xrightarrow{\mathcal{R}} t' \\
(E, \mathcal{R} \cup \Delta \vdash s \rightarrow t) &\Rightarrow (E, \mathcal{R} \cup \Delta \vdash s \rightarrow t') \text{ if } \Delta \vdash t \xrightarrow{\mathcal{R}} t' \\
(E, \mathcal{R} \cup \Delta \vdash s \rightarrow t) &\Rightarrow (E \cup \Delta \vdash s' = t, \mathcal{R}) \text{ if } \Delta \vdash s \xrightarrow{\mathcal{R}} s' \\
&\text{using } \nabla \vdash l \rightarrow r \in \mathcal{R} \text{ and} \\
&l \text{ does not reduce with } \Delta \vdash s \rightarrow t
\end{aligned}$$

As in the case of first-order rewriting, the rule that computes critical pairs should be used in a controlled way, keeping track of the pairs that have already been computed in order to avoid computing the same critical pair repeatedly in the first rule. Different versions of completion use different strategies to apply the rules above. In general, the completion procedure may loop, fail if it reaches a pair  $(E, \mathcal{R})$  where the equations in  $E$  cannot be oriented, and succeed if it reaches an irreducible state  $(\emptyset, \mathcal{R})$  where all the critical pairs in  $\mathcal{R}$  have been computed and the last two transformation rules are not applicable. In the latter case, we say that  $\mathcal{R}$  is the result of the procedure.

**Theorem 2.** *If the completion procedure succeeds for a given set  $E$  of closed equational axioms, obtaining  $\mathcal{R}$  as a result, then  $\mathcal{R}$  is a confluent and terminating nominal rewrite system, and  $\leftrightarrow_{\mathcal{R}}^*$  coincides with the original  $=_E$  (i.e. the set  $\mathcal{R}$ , considered as a set of equations, is equivalent to the set  $E$ ).*

*Proof.* The rewrite system  $\mathcal{R}$  is terminating by Theorem 1, because all the rules have been oriented according to the given ordering, which we are assuming to be well-founded and preserved by substitution and context. It is confluent because it is also locally confluent: since the completion procedure has succeeded, all the critical pairs have been computed and are joinable.

To prove that the symmetric closure of the rewrite relation coincides with  $=_E$ , we proceed as in the first-order case, showing that each step in the derivation  $(E, \emptyset) \Rightarrow^* (\emptyset, \mathcal{R})$  adds a rule which is either an oriented axiom from  $E$  or a consequence of the axioms in  $E$ .

To decide if two terms in context are equal modulo the equational theory generated by a confluent and terminating rewrite system it suffices to check that both terms have the same normal form. Thus, if the completion algorithm succeeds for a given  $E$ , the equational theory generated by  $E$  is decidable.

We end this section with two examples of completion.

*Example 6 (Extracted from [33]).* As ordering take nrpo with any precedence.

$$\begin{array}{l} (\eta) \ a\#X \vdash \lambda([a]app(X, a)) \rightarrow X \\ (\perp) \ \quad \quad \quad app(\perp, Y) \rightarrow \perp \end{array}$$

There is a critical pair between these rules, since  $a\#X, app(X, a) \approx? app(\perp, Y)$  has solution  $\{X \mapsto \perp, Y \mapsto a\}$ :

$$\lambda([a]\perp) = \perp \quad \text{which can be oriented into} \quad \lambda([a]\perp) \rightarrow \perp$$

*Example 7 (Summation).* The following example is an adapted version of the one given in [35]. As ordering take nrpo with precedence  $\Sigma > +$  and  $\Sigma > app > id$ .

$$\begin{array}{l} a\#F \vdash \Sigma(0, [a]app(F, a)) \rightarrow 0 \\ a\#F \vdash \Sigma(s(N), [a]app(F, a)) \rightarrow app(F, s(N)) + \Sigma(N, [a]app(F, a)) \\ id(X) \rightarrow X \\ app(id0, X) \rightarrow id(X) \end{array}$$

With the fourth rule applied on the first and the second we obtain the following two critical pairs

$$\begin{array}{l} \Sigma(0, [a]id(a)) = 0 \\ \Sigma(s(N), [a]id(a)) = app(id0, s(N)) + \Sigma(N, [a]app(id0, a)) \end{array}$$

The obtained equations can be simplified and oriented into

$$\begin{array}{l} \Sigma(0, [a]a) \rightarrow 0 \\ \Sigma(s(N), [a]a) \rightarrow s(N) + \Sigma(N, [a]a) \end{array}$$

The resulting set of rules is closed.

## 5 Conclusions

We have defined two extensions of the rpo to deal with  $\alpha$ -equivalence classes of terms using the nominal approach. The first extension is simple and remains close to the first-order rpo while respecting  $\alpha$ -equivalence. The second extension is  $\alpha$ -compatible and stable under atom-substitution. In both orderings it is possible to orient pairs of terms containing free atoms (although this is not needed to orient rules in closed systems) and variables.

Using the orderings we have designed a completion procedure for nominal rewriting systems, which generalises the Knuth-Bendix procedure for first-order rewriting. This is, to our knowledge, the first completion procedure available for rewrite systems with binding.

As future work, we would like to consider other term orderings, such as the ones defined using polynomial interpretations (see [2, 18]), as well as more sophisticated termination proving techniques like the dependency pair method [1] or the monotonic semantic path ordering [9].

We also plan to investigate richer languages than equational logic. A first natural step is to extend the calculus to Horn or general equational clauses, which can be done by extending the paramodulation inference rule [32] for the nominal case.

## References

1. T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, 236:133-178, 2000.
2. F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, Great Britain, 1998.
3. L. Bachmair, N. Dershowitz and J. Hsiang. Orderings for equational proofs. In *Proc. Symp. Logic in Computer Science*, 346–357, Boston, 1986.
4. L. Bachmair, N. Dershowitz and D. A. Plaisted. Completion Without Failure. In *Resolution of Equations in Algebraic Structures*, vol. 2: Rewriting Techniques, H. Ait-Kaci and M. Nivat, eds., chap. 1, Academic Press, New York, 1989.
5. F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization in the algebraic- $\lambda$ -cube. *Journal of Functional Programming*, 6:613–660, 1997.
6. F. Barbanera and M. Fernández. Intersection type assignment systems with higher-order algebraic rewriting. *Theoretical Computer Science*, 170:173–207, 1996.
7. H. P. Barendregt. *The Lambda Calculus: its Syntax and Semantics (revised ed.)*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1984.
8. F. Blanqui, J.-P. Jouannaud and A. Rubio. The Computability Path Ordering: The End of a Quest. In *Proc. 22nd International Conference Computer Science Logic (CSL) and 17th Annual Conference of the EACSL*. Volume 5213 in *Lecture Notes in Computer Science*. Springer-Verlag, 2008.
9. C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In D. McAllester, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *LNAI*, pages 346–364, Pittsburgh, 2000. Springer-Verlag, 2000.
10. V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 83(1). Elsevier, 1991.
11. V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic confluence. *Information and Computation*, 82:3–28. Elsevier, 1992.
12. C. Calvès. Complexity and implementation of nominal algorithms. PhD thesis, King’s College London, 2010.
13. C. Calvès and M. Fernández. Matching and Alpha-Equivalence Check for Nominal Terms. *Journal of Computer and System Sciences*, Special issue: Selected papers from WOLLIC 2008. Elsevier, 2009.
14. J. Cheney. The complexity of equivariant unification. In *Automata, Languages and Programming, Proceedings of the 31st Int. Colloquium, ICALP 2004*, volume 3142 of *Lecture Notes in Computer Science*. Springer, 2004.
15. R. A. Clouston and A. M. Pitts. Nominal Equational Logic. In *Computation, Meaning and Logic. Articles dedicated to Gordon Plotkin*. L. Cardelli, M. Fiore and G. Winskel (eds.), volume 1496, *Electronic Notes in Theoretical Computer Science*, Elsevier, 2007.
16. R. A. Clouston. Equational logic for names and binders. PhD thesis. University of Cambridge, 2009.
17. N. Dershowitz. Orderings for Term-Rewriting Systems. *Theoretical Computer Science*, vol. 17, no. 3, pp. 279-301. Elsevier, 1982.
18. N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Methods and Semantics*, volume B. North-Holland, 1989.

19. M. Fernández and M.J. Gabbay. Nominal rewriting. *Information and Computation*, Volume 205(6), 2007.
20. M. Fernández and M.J. Gabbay. Closed nominal rewriting and efficiently computable nominal algebra equality. Submitted. Available from [www.dcs.kcl.ac.uk/staff/maribel](http://www.dcs.kcl.ac.uk/staff/maribel).
21. M. Fernández, M.J. Gabbay, and I. Mackie. Nominal rewriting systems. In *Proceedings of the 6th ACM-SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'04), Verona, Italy*. ACM Press, 2004.
22. M. J. Gabbay and A. Mathijssen. Nominal Algebra. *Proceedings of the 18th Nordic Workshop on Programming Theory (NWPT'06)*, 2006.
23. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224. IEEE Computer Society Press, 1999.
24. J.-P. Jouannaud and M. Okada. Executable higher-order algebraic specification languages. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 350–361. IEEE Computer Society Press, 1991.
25. J.-P. Jouannaud and A. Rubio. Polymorphic Higher-Order Recursive Path Orderings. *Journal of ACM*, Vol. 54:1, 2007.
26. Z. Khasidashvili. Expression reduction systems. In *Proceedings of I. Vekua Institute of Applied Mathematics*, volume 36, pages 200–220, Tbilisi, 1990.
27. J.-W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems, introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.
28. D. Knuth and P. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, ed. J. Leech, 263–297. Oxford: Pergamon Press, 1970.
29. K. Kusakari and Y. Chiba. A higher-order Knuth-Bendix procedure and its applications. *IEICE Transactions on Information and Systems*, Vol. E90-D,4, 707–715, 2007.
30. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.
31. M.H.A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942.
32. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 7, pages 372–444. Elsevier Science Publishers and MIT Press, 2001.
33. T. Nipkow and C. Prehofer. Higher-Order Rewriting and Equational Reasoning, In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications. Volume I: Foundations*, Applied Logic Series, volume 8, pages 399–430. Kluwer, 1998.
34. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.
35. J.C. van de Pol. Termination of higher-order rewrite systems. *PhD thesis*, Utrecht University, Utrecht, The Netherlands, December 1996.
36. F. van Raamsdonk. *Confluence and Normalisation for Higher-Order Rewriting*. PhD thesis, Free University of Amsterdam, 1996.
37. Terese. Term Rewriting Systems. Volume 55 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003.
38. C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323:473 – 497, 2004.

## A Appendix: Proofs

The relation  $\bowtie$  is a generalisation of  $\approx_\alpha$  that equates the atoms in  $\mathbb{C}$ . It has similar properties to  $\approx_\alpha$  regarding freshness constraints and permutations:

- Property 5.*
1. If  $\Delta \vdash s \approx_\alpha t$  then  $\Delta \vdash s \bowtie t$ .
  2. If  $\Delta \vdash s \bowtie t$  and  $\Delta \vdash a \# s$  then  $\Delta \vdash a \# t$  or  $a \in \mathbb{C}$ .
  3. If  $\Delta \vdash s \bowtie t$  then  $\Delta \vdash \pi \cdot s \bowtie \pi \cdot t$ , for any  $\pi$  such that  $\forall c \in \mathbb{C}, \pi \cdot c \in \mathbb{C}$  or  $(c \notin \mathbb{A}(s, t) \text{ and } \Delta \vdash c \# s, t)$ .

*Proof.* The first part is proved by induction on the definition of  $\approx_\alpha$ , distinguishing cases according to the last rule used to show  $\Delta \vdash s \approx_\alpha t$ . Most cases are straightforward, since the definition of  $\bowtie$  includes the rules and axioms used to define  $\approx_\alpha$ . The only interesting case is  $(\approx_\alpha \text{absb})$ , where, since  $\mathbb{C}$  is infinite, we can always choose  $c \in \mathbb{C}$  such that  $c \notin \mathbb{A}([a]s, [b]t)$ ; the result then follows by induction.

The second part is proved by induction on  $\bowtie$ . The interesting cases are  $(\bowtie \text{absb})$  and  $(\bowtie \mathbf{X})$ . If the last rule applied was  $(\bowtie \text{absb})$ , the induction hypothesis allows us to conclude, since  $\Delta \vdash a \# s$  implies  $\Delta \vdash \pi \cdot a \# \pi \cdot s$  for any  $\Delta, \pi, a, s$ . If the last rule applied was  $(\bowtie \mathbf{X})$ , then  $s \equiv \pi \cdot X$  and  $t \equiv \pi' \cdot X$ . Since  $\Delta \vdash a \# s$  by assumption, then  $\Delta \vdash \pi^{-1} \cdot a \# X$ . If  $\pi'^{-1} \cdot a = \pi^{-1} \cdot a$  we are done, otherwise,  $\pi'^{-1} \cdot a \in ds(\pi, \pi')$  and therefore either  $\Delta \vdash \pi'^{-1} \cdot a \# X$  and thus  $\Delta \vdash a \# \pi' \cdot X$ , or  $\pi' \cdot (\pi'^{-1} \cdot a) \in \mathbb{C}$  and thus  $a \in \mathbb{C}$ .

The third part is also proved by induction on  $\bowtie$ . We distinguish cases according to the last rule applied:

The case for  $(\bowtie \mathbf{a})$  is trivial, and the cases  $(\bowtie \mathbf{f})$  and  $(\bowtie \text{absa})$  follow directly by induction.

If the last rule applied was  $(\bowtie \mathbf{c})$ , then by assumption  $\pi \cdot s$  and  $\pi \cdot t \in \mathbb{C}$ , hence  $\Delta \vdash \pi \cdot s \bowtie \pi \cdot t$  by  $(\bowtie \mathbf{c})$ .

If the last rule applied was  $(\bowtie \mathbf{X})$ , then we have  $s \equiv \pi' \cdot X$  and  $t \equiv \pi'' \cdot X$ . Since  $ds(\pi \circ \pi', \pi \circ \pi'') = ds(\pi', \pi'')$ , for all  $a \in ds(\pi \circ \pi', \pi \circ \pi'')$  we have  $\Delta \vdash a \# X$  or  $\pi' \cdot a, \pi'' \cdot a \in \mathbb{C}$ . By assumption, the latter implies  $\pi \cdot (\pi' \cdot a) \in \mathbb{C}$  and  $\pi \cdot (\pi'' \cdot a) \in \mathbb{C}$  or  $\pi' \cdot a, \pi'' \cdot a \# \pi' \cdot X, \pi'' \cdot X$ , which implies  $a \# X$ . Thus,  $\Delta \vdash \pi \cdot (\pi' \cdot X) \bowtie \pi \cdot (\pi'' \cdot X)$  by  $(\bowtie \mathbf{X})$ .

If the last rule applied was  $(\bowtie \text{absb})$ , then  $s \equiv [a]s'$  and  $t \equiv [b]t'$  and  $\Delta \vdash (c a) \cdot s' \bowtie (c b) \cdot t'$ , for some fresh atom  $c \in \mathbb{C}$ . If for some  $c' \in \mathbb{C}, \pi \cdot c' \notin \mathbb{C}$  then  $\Delta \vdash c' \# [a]s, [b]t$  by assumption, and since  $c$  is fresh (without loss of generality we can assume  $\pi \cdot c = c$ ), then  $\Delta \vdash c' \# (c a) \cdot s', (c b) \cdot t'$ . By induction,  $\pi \cdot ((c a) \cdot s') \bowtie \pi \cdot ((c b) \cdot t')$  and since  $\pi \cdot ((c a) \cdot s') = (c \pi \cdot a) \pi \cdot s'$  (and similarly for  $\pi \cdot ((c b) \cdot t')$ ), we conclude  $\Delta \vdash \pi \cdot s \bowtie \pi \cdot t$  by  $(\bowtie \text{absb})$ .

Using the properties above, we can prove that  $\bowtie$  is an equivalence relation.

*Property 6.* The relation  $\bowtie$  is reflexive, symmetric and transitive.

*Proof.* Reflexivity and symmetry follow by straightforward induction on the definition of  $\bowtie$ .

Assume  $\Delta \vdash s \bowtie t$  and  $\Delta \vdash t \bowtie u$ . We prove  $\Delta \vdash s \bowtie u$  by induction on the size of  $s$ . We distinguish cases as follows:

- $s \in \mathcal{A}$ : If  $s \equiv a \notin \mathbb{C}$  then  $t, u \equiv a$  and the result follows by  $(\bowtie_{\mathbf{a}})$ . Similarly, if  $s \equiv c \in \mathbb{C}$ , then  $t, u \in \mathbb{C}$ , and the result follows by  $(\bowtie_{\mathbf{c}})$ .
- $s \equiv f(s_1, \dots, s_n)$ : Then also  $t$  and  $u$  are rooted by  $f$ , and the result follows by induction.
- $s \equiv \pi \cdot X$ : Then also  $t \equiv \pi' \cdot X$  and  $u \equiv \pi'' \cdot X$ . If  $a \in ds(\pi, \pi'')$  then  $a \in ds(\pi, \pi')$  or  $a \in ds(\pi, \pi'')$  and the assumptions on  $ds(\pi, \pi')$ ,  $ds(\pi', \pi'')$  allow us to conclude that  $\Delta \vdash s \bowtie u$ .
- $s \equiv [a]s'$ : Then  $t$  and  $u$  are also abstractions. We distinguish cases according to the atoms abstracted in  $t$  and  $u$ .
  - If  $t \equiv [a]t'$  and  $u \equiv [a]u'$  then we conclude directly by induction.
  - If  $t \equiv [a]t'$  and  $u \equiv [b]u'$ , then  $\Delta \vdash s' \bowtie t'$  and  $\Delta \vdash (a \ c) \cdot t' \bowtie (b \ c) \cdot u'$  for some fresh  $c \in \mathbb{C}$ . Without loss of generality, we can assume that  $c$  is also fresh for  $s'$  and then we can use Property 1 (part 3) to derive  $\Delta \vdash (a \ c) \cdot s' \bowtie (a \ c) \cdot t'$ . By induction,  $\Delta \vdash (a \ c) \cdot s' \bowtie (b \ c) \cdot u'$ , and we conclude using  $(\bowtie_{\mathbf{absab}})$ .
  - If  $t \equiv [b]t'$  and  $u \equiv [b]u'$  we proceed as above.
  - If  $t \equiv [b]t'$  and  $u \equiv [d]u'$  then  $\Delta \vdash (a \ c) \cdot s' \bowtie (b \ c) \cdot t'$  for some fresh  $c \in \mathbb{C}$ . Without loss of generality, we can assume that the same  $c$  is fresh for  $[d]u'$  too, and  $\Delta \vdash (b \ c) \cdot t' \bowtie (d \ c) \cdot u'$ . By induction,  $\Delta \vdash (a \ c) \cdot s' \bowtie (d \ c) \cdot u'$ , and we conclude using  $(\bowtie_{\mathbf{absab}})$ .