FROM NOMINAL TO HIGHER-ORDER REWRITING
AND BACK AGAIN

JESÚS DOMÍNGUEZ AND MARIBEL FERNÁNDEZ

Department of Informatics, King’s College London, Strand WC2R 2LS, UK
e-mail address: jesus.dominguez_alvarez@kcl.ac.uk

Abstract. We present a translation function from nominal rewriting systems (NRSs) to combinatory reduction systems (CRSs), transforming closed nominal rules and ground nominal terms to CRSs rules and terms, respectively, while preserving the rewriting relation. We also provide a reduction-preserving translation in the other direction, from CRSs to NRSs, improving over a previously defined translation. These tools, together with existing translations between CRSs and other higher-order rewriting formalisms, open up the path for a transfer of results between higher-order and nominal rewriting. In particular, techniques and properties of the rewriting relation, such as termination, can be exported from one formalism to the other.

1. Introduction

Programs and logical systems often include binding operators. Term rewriting systems \[1, 4\], in their standard form, do not provide support for reasoning on binding structures. This motivated the study of combinations of first-order rewriting systems with the \(\lambda\)-calculus \(2\), which offers a notion of variable binding and substitution. Combinatory reduction systems (CRSs) \(22, 24\) are well-known examples of higher-order rewriting formalisms, where a meta-language based on the untyped \(\lambda\)-calculus was incorporated to a first-order rewriting framework. Other approaches followed, such as HRSs \(28\) and ERSs \(21, 18\) for example.

Techniques to prove confluence and termination of higher-order rewriting systems were studied in \(27, 24, 19\) amongst others. However, the syntax and type restrictions imposed on rules in these systems have prevented the design of completion procedures for higher-order rewriting systems \(29\).

More recently, the nominal approach \(17, 30\) has been used to design rewriting systems with support for binding \(12\). Nominal rewriting systems do not rely on the \(\lambda\)-calculus, instead, two kinds of variables are used: *atoms*, which can be abstracted but behave similarly to constants thus allowing explicit manipulation, and meta-level variables or just *variables*, which are first-order in that they cannot be abstracted and substitution does not avoid capture of unabstracted atoms. On nominal terms \(32, 12\) \(\alpha\)-equivalence is axiomatised using bijective mappings on atoms, known as *permutations*, and a *freshness relation* between atoms and terms. Nominal syntax enjoys many useful properties, for instance, unification modulo \(\alpha\)-equivalence is decidable and unitary \(33\) and nominal matching is linear \(5\).
Nominal rewriting can be implemented efficiently if we use closed rules (roughly speaking, rules that prevent abstracted atoms from becoming free during reductions — a natural restriction, which is also imposed on CRSs, HRSs and ERSs by definition).

The availability of efficient algorithms to solve unification problems on nominal terms motivated the study of the relationship between higher-order and nominal syntax in a series of papers \cite{6,14,26}. In this paper, we focus on the relationship between nominal and higher-order rewriting, specifically between NRSs and CRSs. Whereas the translations provided in \cite{6,26} preserve the unifiability relation, ours preserves the rewriting relation, which is key to the translation of properties such as confluence and termination. We define a translation function from closed NRS rules and ground nominal terms to CRS rules and terms, preserving the rewriting relation. Then, we give a translation function from CRSs to NRSs, improving over a previous translation described in \cite{14}. Since we now have reduction-preserving translations in both directions, properties and techniques developed for one formalism can be exported to the other (e.g., termination techniques based on the construction of a well-founded reduction ordering). A Haskell implementation of the translation functions, along with a tool to prove termination using the nominal recursive path ordering \cite{15}, are available from \cite{10,9}.

Related work. CRSs, HRSs and ERSs are well-known examples of higher-order rewriting formalisms. A comparison of various higher-order rewriting formalisms, with many interesting examples, is provided in \cite{35}; see also \cite{20} for a concise presentation of higher-order rewrite systems. In \cite{34,18}, CRSs are compared with HRSs and ERSs respectively, and in \cite{3} CRSs are expressed in terms of the $\rho$-calculus \cite{7,8}. In \cite{25}, a termination-preserving translation between Algebraic Functional Systems and other higher-order formalisms is presented. Although in this paper we focus on the relationship between NRSs and CRSs, thanks to the existing translations between CRSs and other higher-order rewriting formalisms, this is sufficient to obtain a bridge between nominal and higher-order rewriting.

Our work is closely related to the work reported in \cite{6,26}: Cheney \cite{6} represented higher-order unification as nominal unification, and Levy and Villaret \cite{26} transformed nominal unification into higher-order unification, providing a translation that preserves unifiers. Our translation differs from \cite{6,26} since our requirement is to have a mapping of NRS terms and rules to CRS meta-terms and rules in such a way that reductions are preserved.

This paper is an updated and extended version of \cite{11}. We have included here, in addition to the translation from NRSs to CRSs given in \cite{11}, all the proofs previously omitted due to space constraints as well as a translation from CRSs to NRSs, improving a previous result given in \cite{14}. We provide detailed explanations, and illustrate the translations with examples.

Overview of the paper. The rest of the paper is organised as follows. In section 2 we recall both formalisms, CRSs and NRSs, as defined in \cite{23} and \cite{12} respectively. In section 3 we describe in detail the translation of nominal terms to CRS meta-terms, while in section 4 we extend it to take into account rules and substitution. In Section 5 we prove that nominal rewrite steps can be simulated in CRSs via the translation function. Section 6 presents a translation from CRSs to NRSs. In section 7 we show examples of application of the translations. Section 8 concludes and discusses future work.
2. Preliminaries

We start by briefly recalling the main concepts of nominal rewrite systems and combinatorial reduction systems — two rewriting formalisms that extend the syntax of first-order terms and the notion of rewriting, to facilitate the specification of systems with binding operators. We refer the reader to [24, 12] for more details and examples.

2.1. Nominal Rewriting. A nominal signature Σ is a set of term-formers, or function symbols, f, g, . . . , each with a fixed arity. Fix a countably infinite set X of variables ranged over by X, Y, Z, . . . , and a countably infinite set A of atoms ranged over by a, b, c, . . . , and assume that Σ, X and A are pairwise disjoint. We follow The Permutative Convention [16] Convention 2.3] notation where a, b, c, . . . range over distinct atoms unless stated otherwise. A swapping is a pair of atoms, written (a b). Permutations π are bijections on A such that the set of atoms for which a ̸= π(a) is finite; this is called the support of π, written as support(π). Permutations are represented by lists of swappings, Id denotes the identity permutation. We write π−1 for the inverse of π and π ◦ π′ for the composition of π′ and π. For example, if π = (a b)(b c) and π(a) = b, then π−1 = (b c)(a b) and π−1(a) = c.

Nominal terms, or just terms, are generated by the grammar

\[ s, t ::= a \mid \pi \cdot X \mid [a]s \mid fs \mid (s_1, \ldots, s_n) \]

and called, respectively, atoms, moderated variables or simply variables, abstractions, function applications (which must respect the arity of the function symbol) and tuples (if \( n = 0 \) or \( n = 1 \) we may omit the parentheses). We abbreviate Id \cdot X as X if there is no ambiguity. An abstraction \([a]t\) is intended to represent t with a bound, we say t is in the scope of [a]. Call occurrences of a abstracted if they are in the scope of an abstraction, and unabstracted (or free) otherwise. For example, \( f(X, (a b) \cdot X) \) is a nominal term, and so is \( f([a]X, [b]b) \). The latter term has X in the scope of [a] and b in the scope of [b]. For more examples, we refer the reader to [32, 12].

Definition 2.1. The functions \( V(t) \) and \( A(t) \) are used to compute the sets of variables and atoms in a term \( t \), respectively. They are inductively defined as follows:

\[
\begin{align*}
V(a) &= \emptyset & V([a]t) &= V(t) & V(\pi \cdot X) &= \{X\} \\
V(fs) &= V(s) & V((s_1, \ldots, s_n)) &= V(s_1) \cup \ldots \cup V(s_n) \\
A(a) &= \{a\} & A([a]t) &= A(t) \cup \{a\} & A(\pi \cdot X) &= \text{support}(\pi) \\
A(fs) &= A(s) & A((s_1, \ldots, s_n)) &= A(s_1) \cup \ldots \cup A(s_n)
\end{align*}
\]

Notice that function \( V(t) \) has a syntactic notion of occurrence whereas \( A(t) \) has a logical one because of the permutations suspended on variables. Therefore, \( A(f(a, [b]c)(d c) \cdot X)) = \{a, b\} \).

Ground terms have no variables: \( V(t) = \emptyset \).

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

\[
\begin{align*}
(a b) \cdot a &= b & (a b) \cdot b &= a & (a b) \cdot c &= c \\
(a b) \cdot (\pi \cdot X) &= ((a b) \circ \pi) \cdot X & (a b) \cdot [c]t &= [(a b) \cdot c](a b) \cdot t \\
(a b) \cdot ft &= f(a b) \cdot t & (a b) \cdot (t_1, \ldots, t_n) &= ((a b) \cdot t_1, \ldots, (a b) \cdot t_n)
\end{align*}
\]

Recall atoms a, b, c are considered distinct among them.
The semantics of nominal terms is defined using nominal sets with finite support [31]. A Perm($A$)-set is a set $T$ equipped with a permutation action, such that $\text{Id}: x = x$ and $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$ for each object $x \in T$. A set $S$ of atoms supports $x \in T$ if for all atoms $a, b \notin S$, $(a \ b) \cdot x = x$. A nominal set is a Perm($A$)-set where each element has finite support. In the case of nominal terms, since variables have finite support, we can define the minimal supporting set for a term $t$, written $\text{supp}(t)$, as follows:

\[
\text{supp}(a) = \{a\} \quad \text{supp}(\pi \cdot X) = \pi \cdot \text{supp}(X) \\
\text{supp}(ft) = \text{supp}(t) \quad \text{supp}(\alpha t) = \text{supp}(t) - \{a\} \\
\text{supp}(\langle t_1, \ldots, t_n \rangle) = \text{supp}(t_1) \cup \ldots \cup \text{supp}(t_n)
\]

Substitutions are generated by the grammar: $\sigma ::= \text{Id} \mid [X \mapsto s] \sigma$. We use the same notation for the identity substitution and permutation, and also for composition, since there will be no ambiguity.

Substitutions act on variables, without avoiding capture of atoms. The action of a substitution on a variable is denoted as $\sigma(X)$. Then, the domain of a substitution $\sigma$, $\text{dom}(\sigma)$, is the set of variables such that $\sigma(X) \neq X$ for all $X \in \mathcal{X}$. Write $t \sigma$ for the application of $\sigma$ on $t$, defined as follows: $t \sigma = t$, $t[X \mapsto s] \sigma = (t[X \mapsto s]) \sigma$, and $a[X \mapsto s] = a \ (\pi \cdot X)[X \mapsto s] = \pi \cdot s \ (\pi \cdot Y)[X \mapsto s] = \pi \cdot Y \ (X \neq Y)$

\[
\begin{align*}
& a[X \mapsto s] = a \\
& (\pi \cdot X)[X \mapsto s] = \pi \cdot s \\
& \sigma_t[\pi \cdot Y] = [\pi \cdot Y][X \mapsto s] = \pi \cdot Y \ (X \neq Y)
\end{align*}
\]

\[
\begin{align*}
& \{t_1, \ldots, t_n\}[X \mapsto s] = \{t_1[X \mapsto s], \ldots, t_n[X \mapsto s]\}
\end{align*}
\]

**Definition 2.2** (Freshness constraints). A freshness constraint is a pair $a \# t$ of an atom and a term. A freshness context (ranged over by $\Delta, \nabla, \Gamma$), is a set of constraints of the form $a \# X$. Freshness judgements, written $\Delta \vdash a \# t$, are derived using the rules below.

\[
\begin{align*}
\frac{\Delta \vdash \pi^{-1} a \# X \in \Delta}{\Delta \vdash a \# \pi \cdot X} & \quad (\#X) \\
\frac{\Delta \vdash a \# s, \Delta \vdash a \# t}{\Delta \vdash a \# [s, t]} & \quad (\#\text{tupl}) \\
\frac{\Delta \vdash a \approx \alpha a}{\Delta \vdash a \approx \alpha [a] t} & \quad (\approx \alpha [a])
\end{align*}
\]

In nominal languages one is interested on those terms $t$ that have finite support, because for them there exists always a fresh atom $a$ such that $a \# t$ (recall the set of atoms $\mathcal{A}$ is infinite) [31].

**Definition 2.3** ($\alpha$-equivalence constraints). An $\alpha$-equivalence constraint is a pair $s \approx \alpha t$ of terms where $\approx \alpha$ is a congruence [12]. Equivalence judgements, written $\Delta \vdash s \approx \alpha t$, are derived using the rules below, where $\text{ds}(\pi, \pi') = \{a \mid \pi \cdot a \neq \pi' \cdot a\}$ (difference set).

\[
\begin{align*}
\frac{\Delta \vdash t \approx \alpha t}{\Delta \vdash s \approx \alpha t} & \quad (\approx \alpha t) \\
\frac{\Delta \vdash s \approx \alpha t}{\Delta \vdash s \approx \alpha [a] t} & \quad (\approx \alpha [a])
\end{align*}
\]

Let $P_i$ be a freshness or $\alpha$-equality constraint (for $1 \leq i \leq n$). We write $\Delta \vdash P_1, \ldots, P_n$ when proofs of $\Delta \vdash P_i$ exist (for $1 \leq i \leq n$), using the derivation rules above.
Example 2.4. We can derive $[a](a \ b)\cdot X \approx_\alpha [b]X$ from assumption $a\#X$, using the fact that $(a \ b)(a \ b) = 1d$.

$$
\begin{array}{c}
\frac{ds((a \ b)(a \ b), 1d) = \emptyset}{(a \ b)(a \ b)\cdot X \approx_\alpha X}^{(\approx_\alpha X)} \\
\frac{a\#X}{b\#(a \ b)\cdot X}^{(#X)} \\
\frac{[a](a \ b)\cdot X \approx_\alpha [b]X}{\approx_\alpha [b]X}
\end{array}
$$

**Definition 2.5** (Nominal Rewrite System). A nominal rewrite rule $R = \triangledown \vdash l \to r$ is a tuple of a freshness context $\triangledown$ and terms $l$ and $r$ such that $V(r) \cup V(\triangledown) \subseteq V(l)$.

A nominal rewrite system (NRS) is an equivariant set $\mathcal{R}$ of nominal rewrite rules, that is, a set of nominal rules that is closed under permutations. We shall generally equate a set of rewrite rules with its equivariant closure.

**Definition 2.6.** We extend the notion of occurrence given in Definition 2.1 for both variables, $V(t)$, and atoms, $A(t)$, to include rules, contexts, substitutions, etc. Particularly, for contexts we mean $A(\Delta) = \{a \mid a\#X \in \Delta\}$ for some $X$ and for substitutions, $A(\sigma) = \{A(\sigma(X)) \mid X \in \text{dom}(\sigma)\}$

Example 2.7. The following rules are used to compute prenex normal forms in first-order logic. The signature has term-formers $\text{forall}, \text{exists, not, and}$. Intuitively, equivariance means that the choice of atoms in rules is not important (see [12] for more details), therefore we could change $a$ to $b$ for instance.

$$
\begin{align*}
\text{a\#P} \vdash \text{and}(P, \text{forall}([a]Q)) & \to \text{forall}([a]\text{and}(P, Q)) \\
\text{a\#P} \vdash \text{and}(\text{forall}([a]Q), P) & \to \text{forall}([a]\text{and}(P, Q)) \\
\text{a\#P} \vdash \text{or}(P, \text{forall}([a]Q)) & \to \text{forall}([a]\text{or}(P, Q)) \\
\text{a\#P} \vdash \text{or}(\text{forall}([a]Q), P) & \to \text{forall}([a]\text{or}(P, Q)) \\
\text{a\#P} \vdash \text{and}(P, \text{exists}([a]Q)) & \to \text{exists}([a]\text{and}(P, Q)) \\
\text{a\#P} \vdash \text{and}(\text{exists}([a]Q), P) & \to \text{exists}([a]\text{and}(P, Q)) \\
\text{a\#P} \vdash \text{or}(P, \text{exists}([a]Q)) & \to \text{exists}([a]\text{or}(P, Q)) \\
\text{a\#P} \vdash \text{or}(\text{exists}([a]Q), P) & \to \text{exists}([a]\text{or}(Q, P)) \\
\text{not}(\text{exists}([a]Q)) & \to \text{forall}([a]\text{not}(Q)) \\
\text{not}(\text{forall}([a]Q)) & \to \text{exists}([a]\text{not}(Q))
\end{align*}
$$

Nominal rewriting [12] operates on ‘terms-in-contexts’, written $\Delta \vdash s$ or just $s$ if $\Delta = \emptyset$. Below, $C[\cdot]$ varies over terms with exactly one occurrence of a distinguished variable $1d$-, or just -. We write $C[s]$ for $C[\cdot \mapsto s]$, and $\Delta \vdash \triangledown \theta$ for $\{\Delta \vdash a\#X\theta \mid a\#X \in \triangledown\}$.

**Definition 2.8** (Nominal Rewriting). A term $s$ rewrites with $R = \triangledown \vdash l \to r$ in $\Delta$, written $\Delta \vdash s \to_R t$ (as usual, we assume $V(R) \cap (V(\Delta) \cup V(s)) = \emptyset$), if $s = C[s']$ and there exists $\theta$ such that $\Delta \vdash \triangledown \theta$, $\Delta \vdash l \theta \approx_\alpha s'$ and $\Delta \vdash C[r\theta] \approx_\alpha t$. Since $\Delta$ does not change during rewriting, a rewriting derivation is written $\Delta \vdash s_1 \to_R s_2 \to_R \ldots \to_R s_n$, abbreviated as $\Delta \vdash s_1 \to^* s_n$.

When rules are closed, nominal rewriting can be efficiently implemented using nominal matching (there is no need to consider equivariance). We define closed rewriting below, after defining closed terms.

Closed terms are, roughly speaking, terms without unabstracted atoms, and such that variables behave uniformly with respect to their support. We give a definition below.

**Definition 2.9** (Closedness). A term-in-context $\Delta \vdash t$ is closed if
(1) every atom \( a \in A(t) \) is either an abstraction of \( a, [a] \), or is in the scope of \([a]\);

(2) if \( \pi \cdot X \) occurs in the scope of an abstraction of \( \pi \cdot a \) then any occurrence of \( \pi' \cdot X \) occurs in the scope of an abstraction of \( \pi' \cdot a \) or \( a\#X \in \Delta \);

(3) for any pair \( \pi_1 \cdot X, \pi_2 \cdot X \) occurring in \( t \), and \( a \in ds(\pi_1, \pi_2) \), if \( a \) does not occur in the scope of an abstraction in one of the occurrences then \( a\#X \in \Delta \).

A rewrite rule \( \nabla \vdash l \rightarrow r \) is closed if \( \nabla \vdash (l, r) \) is a closed term.

The first condition in the definition specifies that no atom can occur free in a closed term. The second condition states that if an atom \( a \) in an instance of a variable \( \pi \cdot X \) is captured (i.e., \( \pi \cdot X \) is under an abstraction for \( \pi \cdot a \)) then it is captured in all occurrences of \( X \) or it is fresh for \( X \). The third condition says that if two occurrences of \( X \) have different suspended permutations, then any atom in the difference set that could occur in an instance of \( X \) is captured.

For example, \([a]f(X, a)\) is closed, but \( f(X, a) \) and \( f(X, [a]X) \) are not, however \( a\#X \vdash f(X, [a]X) \) is closed. All the rewrite rules in Example \ref{ex:rewrites} are closed.

Closedness can be easily checked using the nominal matching algorithm \cite{DBLP:journals/tcs/Dolan02}, as follows. First, given a term in context \( \nabla \vdash t \), or more generally, a pair \( P = \nabla \vdash (l, r) \) (this could be a rule \( R = \nabla \vdash l \rightarrow r \)), let us write \( P^\sigma = \nabla^\sigma \vdash (l^\sigma, r^\sigma) \) to denote a freshened variant of \( P \), i.e., a version where the atoms and variables have been replaced by ‘fresh’ ones. We shall always explicitly say what \( P^\sigma \) is freshened for when this is not obvious. For example, a freshened version of \( (a\#X \vdash f(X) \rightarrow X) \) with respect to itself and to \( a'\#X \vdash a' \) is \( (a''\#X' \vdash f(X') \rightarrow X') \). We will write \( A(P')\#V(P) \) to mean that all atoms mentioned in \( P' \) are fresh for each of the variables occurring in \( P \). Let \( \nabla^\sigma \vdash t^\sigma \) be a freshened version of \( \nabla \vdash t \). Then \( \nabla \vdash t \) is closed if there exists a substitution \( \sigma \) such that \( \nabla, A(\nabla^\sigma \vdash t^\sigma)\#V(\nabla \vdash t) \vdash \nabla^\sigma \sigma \approx_\alpha t \). A similar check can be done for nominal rewrite rules.

**Definition 2.10** (Closed Rewriting). Let \( R^\sigma \) be a freshened version of the rule \( R \) with respect to \( \Delta, s, t \) (i.e., a version where the atoms and variables in \( R \) have been replaced by fresh ones; as shown in \cite{DBLP:conf/csl/Dolan06}, it does not matter which particular freshened \( R^\sigma \) we choose). We write \( \Delta \vdash s \rightarrow^R t \) if \( \Delta, \Delta' \vdash s \rightarrow^R t \), where \( \Delta' = A(R^\sigma)\#V(\Delta, s) \), and call this a closed rewriting step. The subindex \( R \) may be omitted if it is clear from the context.

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 the system is locally confluent \cite{DBLP:journals/tcs/Dolan02}.

Notice it is possible to do (standard) rewriting with a closed rule, closed rewriting with a (standard) rule, or closed rewriting with a closed rule.

**Example 2.11.** We show an application of closed rewriting on the term

\[
\vdash \text{and}(X, \forall a([b]f(b)))
\]

using the first rule in Example \ref{ex:rewrites}

\[
a'\#X \vdash \text{and}(X, \forall a([b]f(b))) \rightarrow \forall a([a']\text{and}(X, f(a')))\]

To generate it, we first obtain a freshened variant of the rule with respect to itself and the given term:

\[
a'\#P' \vdash \text{and}(P', \forall a'[Q']) \rightarrow \forall a'([a']\text{and}(P', Q')).
\]

Then, the matching algorithm produces a substitution \( \theta = [P' \mapsto X][Q' \mapsto f(a')] \), and \( a'\#P'\theta \) holds.
2.2. Combinatory Reduction Systems. A combinatory reduction system [22, 24] is a pair consisting of an alphabet $A$ and a set of rewrite rules.

The alphabet consists of: a countably infinite set $V$ of variables ranged over by $a, b, c, \ldots$; a countably infinite set $MV$ of meta-variables with fixed arities, written as $Z^n_i$ where $n$ is the arity of $Z^n_i$ (when $n = 0$ we omit the parentheses); function symbols $f, g, \ldots$ with fixed arities; and an abstraction operator $[\cdot]$. Only variables can be abstracted. We write $MV(t)$ (resp. $Var(t)$) for the set of meta-variables (resp. variables) occurring in a term $t$ (the same notation is used for rules, etc.).

In CRSs a distinction is made between meta-terms and terms. Meta-terms are the expressions built from the symbols in the alphabet, in the usual way. Variables that occur in the scope of the abstraction operator are bound, and free otherwise. Meta-terms are defined modulo renaming of bound variables, that is, a meta-term represents an $\alpha$-equivalence class. Terms are meta-terms that do not contain meta-variables, and are also defined modulo $\alpha$-equivalence. A (meta-)term is closed if every variable occurrence is bound. CRSs adopt the usual naming conventions (also known as Barendregt’s variable conventions): in particular, all bound variables are chosen to be different from the free variables.

A rewrite rule is a pair of meta-terms, written $l \Rightarrow r$, where $l, r$ are closed, $l$ has the form $f(s_1, \ldots, s_n)$ where $n \geq 0$ (when $n = 0$ we omit the parentheses), $MV(r) \subseteq MV(l)$, and $MV(l)$ occur only in the form $Z^n_i(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n$ are pairwise distinct bound variables. We call this form a meta-application.

Example 2.12. The $\beta$-reduction rule for the $\lambda$-calculus is written:

$$\text{app}(\text{lam}([a]Z(a)), Z') \Rightarrow Z(Z')$$

where $Z$ is a unary meta-variable and $Z'$ is 0-ary.

The reduction relation is defined on terms. To extract from rules the actual rewrite relation, each meta-variable is replaced by a special kind of $\lambda$-term, and in the obtained term all $\beta$-redexes and the residuals of these $\beta$-redexes are reduced (i.e. a complete development is performed). Formally, the rewrite relation is defined using substitutes and valuations. An $n$-ary substitute is an expression of the form $\lambda a_1 \ldots a_n.t$, where $t$ is a term and $a_1, \ldots, a_n$ are different variables. An $n$-ary substitute can be applied to a $n$-tuple $s_1, \ldots, s_n$ of terms, and the result is the term $t$ where $a_1, \ldots, a_n$ are simultaneously replaced by $s_1, \ldots, s_n$. A valuation $\sigma$ is a map that assigns an $n$-ary substitute to each $n$-ary meta-variable. It is extended to a mapping from meta-terms to terms: given a valuation $\sigma$ and a meta-term $t$, first we replace in $t$ all meta-variables by their images in $\sigma$ and then we perform the developments of the $\beta$-redexes created.

A context is a term with an occurrence of a special symbol $[]$ called hole. A rewrite step is now defined in the usual way: if $l \Rightarrow r$ is a rewrite rule, $\sigma$ a valuation and $C[]$ a context, then $C[l\sigma] \Rightarrow C[r\sigma]$.

Example 2.13. The following is a rewrite step using the $\beta$-rule given in Example 2.12

$$\text{app}(\text{lam}([a]f(a, a)), t) \Rightarrow_\beta f(t, t).$$

To generate it we use the valuation $\sigma$ that maps $Z$ to $\lambda(b).f(b, b)$ and $Z'$ to the term $t$. Then $\text{app}(\text{lam}([a]Z(a)), Z')\sigma$ is the term $\text{app}(\text{lam}([a]f(a, a)), t)$ obtained by first replacing $Z$ and $Z'$ as indicated by $\sigma$ and then reducing the $\beta$-redex $(\lambda(b).f(b, b))(a)$. Also, $Z(Z')\sigma$ is the term $f(t, t)$ obtained by first replacing $Z$ and $Z'$, resulting in $(\lambda(b).f(b, b))(t)$, and then $\beta$-reducing to $f(b, b)\{b \mapsto t\}$. 

From Nominal to Higher-Order Rewriting and Back Again
2.3. Symmetric groups. The following definitions and proofs will be useful later, in Section 6 when translating CRSs into NRSs.

**Definition 2.14.** The symmetric group \( S_n \) is the group of bijections (permutations) of \( \{a_1, \ldots, a_n\} \) to itself. A standard notation for the permutation that sends \( i \to \pi(i) \) is the two-line notation or array form

\[
\begin{pmatrix}
a_1 & a_2 & a_3 & \ldots & a_n \\
\pi(a_1) & \pi(a_2) & \pi(a_3) & \ldots & \pi(a_n)
\end{pmatrix}
\]

Under composition of mappings, the permutations of \( \{a_1, \ldots, a_n\} \) are a group.

A permutation \( \pi \in S_n \) is a \( k \)-cycle if there are distinct elements \( a_1, a_2, \ldots, a_k \) such that \( \pi(a_1) = a_2, \pi(a_2) = a_3, \ldots, \pi(a_k) = a_1 \) and \( \pi \) fixes every other element. A 2-cycle permutation is known as a transposition. There is a standard notion for \( k \)-cycle forms:

\[(a_1, a_2, a_3, \ldots, a_k)\]

A pair of cycles \( (a_1, \ldots, a_n) \) and \( (a'_1, \ldots, a'_n) \) are disjoint when the sets \( \{a_1, \ldots, a_n\} \) and \( \{a'_1, \ldots, a'_n\} \) are disjoint.

**Lemma 2.15.**
- Every permutation is uniquely expressible as a product of disjoint cycles.
- Disjoint cycles commute.

**Proof.** Well-known

**Theorem 2.16 (Product of Transpositions).** Every permutation in \( S_n \), \( n > 1 \), can be expressed as a product of 2-cycles.

**Proof.** Well-known

**Definition 2.17 (Decomposing a \( k \)-cycle into a product of transpositions).** A \( k \)-cycle \((a_1, a_2, a_3, \ldots, a_{k-1}, a_k)\) in \( S_n \) can be decomposed into transpositions as follows:

\[(a_1, a_2, a_3, \ldots, a_{k-1}, a_k) = (a_1 a_k)(a_1 a_{k-1}) \ldots (a_1 a_3)(a_1 a_2)\]

following the grammar of permutations given in Section 2.1.

Using this method of decomposing \( k \)-cycles we can easily decompose any permutation by first writing the permutation as a product of disjoint cycles, and then decomposing each cycle into 2-cycles as shown in the example below.

**Example 2.18.** The following partial bijective function:

\[
\begin{align*}
f(a) &= c & f(b) &= d & f(c) &= e \\
f(d) &= f & f(e) &= g & f(f) &= h \\
f(g) &= a & f(h) &= b & f(i) &= i
\end{align*}
\]

has the following array form associated with it.

\[
f = \begin{pmatrix}
a & b & c & d & e & f & g & h & i \\
c & d & e & f & g & h & a & b & i
\end{pmatrix}
\]

In this case, the first row represents the elements in the domain, in lexicographic order, and the second row their respective image.

To convert array form notation into cycle notation we follow these steps:
• Start with the smallest letter in the set, in this case \( a \), since \( f(a) = c \) we begin the cycle by writing
\[
(a, c, \ldots)\ldots
\]
Notice we could start with any letter since there are a number of equivalent representations of \( f \) in cycle form. Non-unique representation does not alter the action of the permutations in \( f \).

• Next, \( c \) maps to \( e \), so we continue building the cycle
\[
(a, c, e, \ldots)\ldots
\]

• Continuing in this way we construct \( (a, c, e, g, \ldots)\ldots \) and since \( g \) maps back to \( a \), then we close off the cycle
\[
(a, c, e, g)\ldots
\]

• Next, we pick the smallest letter that does not appear in any previous constructed cycle, this is letter \( b \) in this case, and repeat the previous pattern to construct a new cycle:
\[
(a, c, e, g)(b, d, f, h)\ldots
\]

• Finally the last letter \( i \) is picked and the cycle is constructed. In this case \( i \) maps to itself:
\[
(a, c, e, g)(b, d, f, h)(i)
\]

• Decomposition into 2-cycle form by application of Definition 2.17 would look like this:
\[
(h \ b)(f \ b)(d \ b)(g \ a)(e \ a)(c \ a)
\]

where 1-cycle \( (i) \) is discarded by application of Theorem 2.16.

This simple method of converting bijective mappings in array form into a 2-cycle representation of permutations which preserve the action of the mappings is instrumental for a correct translation of meta-applications in CRSs into permutations suspended over variables in NRSs. We postpone further discussion along with the formal definition of the conversion procedure for Section 6 where CRS rules and terms are translated to NRSs.

3. Translating from Nominal to CRS Syntax

In this section, we give an overview of the main issues surrounding the translation between NRS and CRS syntax, along with our approach to solve them. Further examples and formal proofs are given after defining the translation function.

3.1. Overview of the Problem. In order to design a function that transforms NRSs to CRSs, we must take into account the following distinctions between formalisms:

- CRS rules are closed by definition; this is not the case for nominal rules. As a result, the set of NRS rules suitable for translation must be restricted to closed rules.
- CRSs make a distinction between meta-terms and terms; rewriting is defined only on terms. Such a distinction does not occur in NRSs. To solve such issue, nominal rewriting operates only on ground terms.
NRSs contain a (possibly empty) set of freshness conditions to avoid accidental name capture. Such a mechanism does not exist in CRSs, where (meta-)terms are defined modulo $\alpha$. Therefore, freshness conditions must be considered when constructing both the variable argument list part of a meta-application and the list of bindings for a CRS substitute.

• Nominal variables have arity zero whereas CRS meta-variables may have non-zero arity. The arity of the meta-variable is assigned by association to the length of the argument list created by the translation.

• A moderated variable $\pi \cdot X$ contains a suspended permutation $\pi$ which is applied immediately after instantiating $X$. There are no permutations in CRSs. Intuitively, we must observe the potential effects of applying the permutation to any instantiation of $X$ and translate accordingly to CRS in order to simulate the action of $\pi$.

• We must simulate first-order substitution in CRSs, allowing variable capture. To that extent, the algorithm must recognize abstractions and build binding lists suitable to replicate variable capture in CRSs.

To start with, we consider how to simulate first-order substitution in CRSs. To this extent an auxiliary function $\Lambda$ is created such that $\Lambda$ traverses a nominal term $t$ and outputs, for each nominal variable $X$ in $t$, a set of distinct atoms that occur abstracted above any of the occurrences of $X$ in $t$, for instance, if $t = g([a][b]X, [a][b]X)$ then $\Lambda_t(X) = \{a,b\}$. Atoms in $\Lambda_t(X)$ could be captured if $X$ is instantiated by a term that contains these atoms free, e.g.: $\sigma(X) = f(a,b)$. Since variable capture must be allowed, $\Lambda_t(X)$ is used to create the variable binding list in a substitute of a meta-variable $X$, in this case, $\Lambda(a.b).f(a,b)$. Moreover, as CRS rules are closed, $\Lambda_t(X)$ also aids on constructing the list of bound variables associated with each occurrence of a meta-variable $X$ (i.e., the arguments of $X$ when instantiated by a CRS substitute). Therefore, the example above would translate to $\hat{t} = g([a][b]X^2(a,b), [a][b]X^2(a,b))$ with index 2 being the length of the created argument list (and left implicit from now onwards).

Additionally, both lists (i.e., arguments and binders) depend on freshness conditions $\Delta$ for the nominal term $t$ such that if $a \notin X \in \Delta$, we do not need to take $a$ into account, that is, $a$ cannot occur free in a substitution $\sigma$ for $X$, $\sigma(X)$, and thus there is no possibility of capturing $a$. However, $a$ still could be part of the CRS translation, this is due to the suspended list of swappings $\pi$ in each occurrence of $X$. Each $\pi$ is applied directly to $\sigma(X)$, generating distinct versions of the same initial substitution. We must take this into account when translating nominal rules.

In our translation, the list of variable arguments occurring in each meta-application is ordered with respect to a total ordering. By doing so, there is no relation between the position of each variable in the argument list of the meta-application and the position of the abstractions in the CRS meta-term. On the other hand, there is a bijection between the variables in the binding list added to the substitute for $X$ and the variables in the argument list, as expected. Furthermore, initially both binding and argument list are equivalent so that each occurrence of a substitute for $X$ generated during the translation is equivalent modulo $\alpha$. This property allow us to choose just one of the occurrences to work with and then apply it back to all the occurrences in the translated CRS term. We use the leftmost occurrence of a substitute for $X$ and also prove that each (possibly distinct) list of

\[1\text{We have chosen a lexicographic ordering but any other total ordering also works.}\]
arguments generated from the translation algorithm preserves closedness in the CRS rules (see Definition 2.9).

Another advantage of generating $\Lambda$ is that, by collecting the information on the total number of abstractions beforehand, we are able to transform any subterm of $t$ without loss of information, generating the same CRS meta-application as when the entire term is translated. This comes in handy when representing rewriting steps. We show a few introductory examples below.

Permutations are the main cause of variations among occurrences of the same variable in a term when instantiated, leading to possible modifications of the binding structure. Consider, for example,

$$t = \vdash f([a][b]X, [a][b](a b)\cdot X)$$

and the substitution $\sigma = [X \mapsto g(a, b)]$, which produces the term

$$t\sigma = f([a][b]g(a, b), [a][b]g(b, a))$$

where atoms $a, b$ have been swapped on the second occurrence of $X$. How should we take into account these bijections in the CRSs syntax?

Intuitively, we apply each $\pi$ directly to the set of atoms $\Lambda_t(X) = \{a, b\}$, for each occurrence of $\pi \cdot X$ in $t$, resulting in two argument lists: $(a, b)$ for the first occurrence of $X$ and $(b, a)$ for the second one. However, this approach is not effective when we encounter occurrences of $\pi \cdot X$ where swappings in $\pi$ contain atoms which do not occur abstracted above $X$, therefore not contained in $\Lambda$. Take for instance the nominal term

$$s = [a](a b)\cdot X$$

where $\Lambda_s(X) = \{a\}$. A direct application of $\pi$ to $\Lambda_s(X)$ results in the CRS meta-term

$$\hat{s} = [a]X(b).$$

We immediately notice two problems with this translation: The possibility of atom $b$ occurring in an instantiation of $X$ has not been accounted for in the CRS translation, since we expect $b$ to be renamed to $a$ by means of the swapping $(a b)$, yet $a$ does not appear in the variable argument list after application of $\pi$ to $\Lambda_s(X)$. Furthermore, if our goal is to translate NRS rules into CRS rules, and CRS rules are closed by definition, the application of $\pi$ to $\Lambda_s(X)$ produces a list of atoms no longer bound above $X$, therefore not contained in $\Lambda$. Take for instance the nominal term

$$\hat{s} = [a]X(b).$$

We immediately notice two problems with this translation: The possibility of atom $b$ occurring in an instantiation of $X$ has not been accounted for in the CRS translation, since we expect $b$ to be renamed to $a$ by means of the swapping $(a b)$, yet $a$ does not appear in the variable argument list after application of $\pi$ to $\Lambda_s(X)$. Furthermore, if our goal is to translate NRS rules into CRS rules, and CRS rules are closed by definition, the application of $\pi$ to $\Lambda_s(X)$ produces a list of atoms no longer bound above $X$. As a direct result, the translated rule is no longer closed in CRS, therefore not valid.

Alternatively, we apply a lexical order to the set $\pi^{-1}(\Lambda_t(X))$ built for any nominal term $t$ and occurrence $\pi \cdot X$ in $t$, obtaining an intermediate list: $xs$. It contains those atoms that could be captured if occurring free in a substitution for $X$, that is, $xs$ is the initial list of binders that allow our translation to capture variables. Next, we apply $\pi$ to $xs$ to return an ordered and filtered version of $\Lambda_t(X)$ as a list, which we call $\mathcal{XS}$. The list $\mathcal{XS}$ is thus finally displayed as the variable argument list for the meta-application of an occurrence of $X$. Therefore, at this point we have a bijection from the atoms in $xs$ to those in $\mathcal{XS}$ such that, $xs_i \mapsto \mathcal{XS}_i$ is the mechanism that maps a bound variable $a \in xs$ at position $i$ to the variable argument $\pi(a) \in \mathcal{XS}$ at position $i$ when a substitution is provided. In addition, $\pi$ is also applied to the nominal substitution $\sigma(X)$ prior translation, to rename atoms in $\text{support}(\pi)$ not in scope of $\Lambda_t(X)$. As a result, $\pi$ must also be applied to the binding list $xs, \pi \cdot xs$, to preserve the binding structure when added to $\pi \cdot \sigma(X)$. Hence $xs$ also becomes $\mathcal{XS}$. 
We provide a more detailed explanation of our algorithm after its definition. Now, we look at the examples again, applying the new approach:

\[
(t = \vdash f (\; a \; | b \; X, \; [a][b](a \; b) \cdot X), \; \sigma = [X \mapsto g(a, b)])
\]

is translated as

\[
(\hat{t} = \vdash f (\; a \; | b \; (a, b), \; [a][b]X(b, a)), \; \hat{\sigma} = [X \mapsto \Lambda(a, b), g(a, b)])
\]

where

\[
\hat{t}\hat{\sigma} = \vdash f (\; [a][b]g(a, b), \; [a][b]g(b, a))
\]

Also,

\[
s = [a](a \; b) \cdot X
\]

translates to the CRS meta-term

\[
\hat{s} = [a]X(a)
\]

which is now closed. And a nominal substitution \(\sigma = [X \mapsto g(a, b)]\) to instantiate nominal term \(s\) would translate into the valuation \(\hat{\sigma} = [X \mapsto \Delta(a, b) \cdot g(a, b)]\).

So far, we have considered the translation of terms in context. Next, we consider rewrite rules.

To ensure nominal rules are translated into CRS rules by the algorithm, we must restrict ourselves to closed nominal rules (see Definition 2.9) since CRS rules use closed meta-terms. In closed nominal terms, atoms only occur in the scope of abstractions, and abstracted atoms are preserved across occurrences of the same variable \(X\) (otherwise they must be declared fresh for \(X\) in \(\Delta\)). Swapping of any atom to a unabstracted atom must also be consistent for each occurrence of \(X\), else a freshness is in \(\Delta\) too. This guarantees the translation algorithm maps nominal variables to meta-variables with fixed arities. We later prove that arities are respected across the term.

We end this section by providing some more examples. Take the nominal rule

\[
(\vdash f (\; a \; | X) \rightarrow X)
\]

This rule is not closed: if \(\sigma(X)\) contains the unabstracted atom \(a\), when \(\sigma\) is applied to both sides of the rule, \(a\) would be abstracted in left-hand side but not on the right-hand side. This rule would translate to

\[
f ([a]X(a)) \Rightarrow X(a)
\]

which is not a valid CRS rule. This illustrates the need to restrict to closed nominal rules. Notice that, by having a function \(\Lambda\) local to each meta-term in the rule, the issue still persists: The translation of the previous nominal rule, where now \(\Lambda f_{\; [a]X}(X) = \{a\}, \Lambda_X(X) = \emptyset\), outputs the rule

\[
f ([a]X(a)) \Rightarrow X
\]

which is not a CRS rule either. In this case, although each side is closed, the arity of \(X\) is not respected, so \(\sigma\) cannot correctly instantiate both occurrences of \(X\). A final attempt to amend this issue would be to declare each occurrence of a meta-variable as distinct, yet this could create new meta-variables on the right hand side, which are not in the left-hand side, deeming the rule invalid.

We have discussed the issues governing the translation of nominal rules to CRS rules along with strategies to solve such issues. The rest of the section formalises this approach and provides examples.
3.2. Translating Nominal Terms. For each nominal signature $\Sigma$, and sets $A$ and $X$ of atoms and variables, we consider a CRS alphabet containing $\Sigma$, variables $A$ and metavariables $X$.

First we define an auxiliary function, $\Lambda$, to compute, for each variable occurring in a nominal term, the set of atoms that may be captured when a nominal variable is instantiated.

Definition 3.1 (Mapping $\Lambda_t$). For each nominal term $t$, we define $\Lambda_t : X \to P(A)$ such that $\Lambda_t(X) = \{a_1, \ldots, a_n\}$ if $X \in V(t)$ has $k$ occurrences in $t$, $A_i$ is the set of atoms abstracted above the $i$th occurrence of $X$, and $\{a_1, \ldots, a_n\} = A_1 \cup \ldots \cup A_k$. In other words, $\Lambda_t(X)$ is the set of all the atoms abstracted above occurrences of $X$ in $t$.

Formally it is defined as follows: Let $t$ be a nominal term. For each $X \in V(t)$, $\Lambda_t(X) = \Lambda' (\varnothing, t)(X)$ where $\Lambda'(\cdot, \cdot)$ is an auxiliary function defined inductively over the structure of $t$ as follows:

- $\Lambda'(A, a)(X) = \varnothing$
- $\Lambda'(A, \pi \cdot X)(X) = A$
- $\Lambda'(A, \pi \cdot Y)(X) = \varnothing$
- $\Lambda'(A, [a]s)(X) = \Lambda'(A \cup \{a\}, s)(X)$
- $\Lambda'(A, f s)(X) = \Lambda'(A, s)(X)$
- $\Lambda'(A, (s_1, \ldots, s_n))(X) = \Lambda'(A, s_1)(X) \cup \ldots \cup \Lambda'(A, s_n)(X)$

Then, $\Lambda$ applied to $t = ([a]X, [b]X, [c]Y)$ resolves to $\Lambda_t(X) = \{a, b\}$ and $\Lambda_t(Y) = \{c\}$.

Next, we define the translation of nominal terms into CRS (meta-)terms.

Definition 3.2 (Term Translation). Let $\Delta \vdash t$ be a nominal term-in-context and $\Lambda_t$ as in Definition 3.1. Then $T(\Delta, t) = [[t]]_{\Lambda_t}$, where $[[\cdot]]_{\Lambda_t}$ is an auxiliary function defined by induction over the structure of nominal terms as follows:

- (atom) $[[a]]_{\Lambda_t} = a$
- (var) $[[\pi \cdot X]]_{\Lambda_t} = X(\pi s)$ where $\pi s \triangleq \pi \cdot xs$ (we omit $\pi s$ if empty) $xs \triangleq \text{toAscList}(\pi^{-1} \cdot \Lambda_t(X) - \{a \mid a \# X \in \Delta\})$
- (abs) $[[[a]s]]_{\Lambda_t} = [a][s]_{\Lambda_t}$
- (fun) $[[fs]]_{\Lambda_t} = [f][s]_{\Lambda_t}$
- (tuple) $[[s_1, \ldots, s_n]]_{\Lambda_t} = ([s_1]_{\Lambda_t}, \ldots, [s_n]_{\Lambda_t})$

where $\text{toAscList}$ is a function that builds a sorted list\(^2\) from a set of atoms.

The interesting case in the translation is that of a variable. Intuitively, the list $xs$ contains the atoms $a_i$ that will be captured if they are free in an instance of $X$, because $\pi \cdot X$ occurs under the scope of an abstraction for $\pi \cdot a_i$. Note that if $a \notin \pi s$ then if $\pi^{-1} \cdot a$ is in a substitution for $X$, then $a$ will be in $\text{supp}(t)$ (it will not be captured). In other words, $a \notin \pi s$ implies that if $\pi^{-1} \cdot a \in \text{supp}(X)$ then $a \in \text{supp}(t)$.

Example 3.3. The nominal term

$$\vdash [a][b]X$$

is translated as the CRS meta-term

$$[a][b]X(a, b)$$

where we include both variables in the meta-application as they may appear free in a substitution $\sigma(X)$. Failure to include them could induce a renaming of the bound variables in

\(^2\)List of atoms in ascending lexical order.
the meta-term in order to avoid variable-capture, as defined in [24], leading to a translation disassociated from its original input.

Freshness constraints also have to be taken into account. Consider the example

\[ a \# X \vdash [a][b]X \]

which is translated to the CRS meta-term

\[ [a][b]X(b) \]

Since \( \sigma(X) \) must satisfy \( \Delta \) (that is, \( a \# \sigma(X) \)), we do not include \( a \) in the meta-application.

However, a freshness constraint does not always discard an atom from inclusion in the list of arguments. Ultimately it depends on the permutations. We adjust our example to show this. Consider

\[ a \# X \vdash [a][b](a\ b)\cdot X \]

In this case we should take into account the mapping \( b \mapsto a \) but not \( a \mapsto b \) since \( a \# X \in \Delta \). Our translation outputs the CRS meta-term

\[ [a][b]X(a) \]

which suggests that \( a \) may occur free in \( \sigma(X) \) contradicting the nominal constraint \( a \# X \in \Delta \). However, since any nominal substitution \( \sigma \) that instantiates \( X \) must also satisfy \( \Delta \), the atom \( a \) does not occur unabstracted in \( \sigma(X) \) or in its CRS translation. Hence the mapping \( a \mapsto b \) is discarded.

**Example 3.4.** The nominal term

\[ \vdash [a][b](a\ c)\cdot X \]

where there are no freshness conditions (\( \Delta = \emptyset \)), has \( \Lambda_t(X) = \{a, b\} \), \( xs = [b, c] \), and \( \overline{x} = [a, b] \). The term also includes a mapping \( a \mapsto c \) from an abstracted atom \( a \) to an unabstracted atom \( c \) that is not taken into account. Our algorithm is designed to translate NRSs into CRSs, which are closed by definition. Accordingly, this particular kind of mapping cannot be explicitly represented at term level thus it is applied to the substitute that instantiates \( X \), if any.

This method is shown in more detail in section [3] when describing the translation of substitutions. Our translation function for terms produces the meta-term

\[ [a][b]X(b, a) \]

which effectively takes into account the rest of the mappings in the permutation, that is, \( b \mapsto b, c \mapsto a \), generating a closed CRS meta-term.

One more example, this time considering distinct permutations on each occurrence of a nominal variable.

**Example 3.5.** The translation of closed nominal term-in-context

\[ t = f \# X \vdash f([a][b][c](c\ f)(a\ b)(d\ e)\cdot X_1, [a][b][c](c\ f)(d\ e)\cdot X_2) \]

is the CRS meta-term

\[ \hat{t} = f([a][b][c]X(b, a), [a][b][c]X(a, b)) \]

where sub-indices are used to differentiate occurrences of same variable \( X \) and:

- \( \Lambda_t(X) = \{a, b, c\} \),
- \( xs_1 = xs_2 = [a, b] \),
- \( \pi_1 \cdot xs_1 = \overline{xs_1} = [b, a] \) and \( \pi_2 \cdot xs_2 = \overline{xs_2} = [a, b] \)
Notice that if atom \( c \) occurs unabstracted as part of an instantiation of \( X, \sigma(X) \), there exists a mapping \( c \mapsto f \) on both permutations that successfully renames \( c \) to \( f \) when translating \( \sigma(X) \) to CRSs. This is formally defined in Definition 5.1.

The following lemma proves that the initial list of atoms denoted as \( xs \) is identical for all occurrences of a variable in the nominal term to be translated.

**Lemma 3.6 (Equivalence).** Let \( \Delta \vdash t \) be a closed term-in-context and \( \mathcal{T}(\Delta, t) = \hat{t} \) its CRS translation. If \( \pi_1 \cdot X \) and \( \pi_2 \cdot X \) are two occurrences of the same variable \( X \) in \( t \), and \( X(x_1), X(x_2) \) are their respective translations in \( \hat{t} \), then \( \pi_1^{-1} \cdot x_1 = \pi_2^{-1} \cdot x_2 \).

**Proof.** This is a consequence of the definition of \( \pi \) and the fact that the term is closed. More precisely, by Definition 3.2, the translation of \( \pi \cdot X \) is \( \pi(x) \) where \( \pi(x) = \pi \cdot x \) and \( x \equiv \text{toAscList}(\pi^{-1}(\sigma(X))) - \{a | a \# X \notin \Delta \} \).

It is sufficient to prove that if an atom \( a \in xs_1 \) then \( a \in xs_2 \), and vice versa.

Now, for any \( a \) such that \( a \in xs_1 \), it is also the case that \( \pi_1(a) \in \Lambda_t(X) \) and \( a \# X \notin \Delta \) by definition of \( xs \), then either

1. \( \pi_2(a) \notin \Lambda_t(X) \), or
2. \( \pi_2(a) \in \Lambda_t(X) \), thus \( a \in xs_2 \).

No other cases are possible.

In case (1), any substitution of \( X \) containing atom \( a \) free is in the scope of an abstraction in \( \pi_1 \cdot X \) since \( \pi_1(a) \in \Lambda_t(X) \), but unabstracted under \( \pi_2 \cdot X \), since \( \pi_2(a) \notin \Lambda_t(X) \). Since the term is closed it must be the case that \( a \# X \in \Delta \) by Definition 2.9, contradicting the fact that \( a \in xs_1 \). Hence it is the case that \( \pi_2(a) \in \Lambda_t(X) \) too, as stated in (2). Thus, we have established that for each \( a \) in \( xs_1 \) at some position \( i \), \( a \in xs_2 \) at some position \( j \). Similarly, we can prove \( a \in xs_2 \) implies \( a \in xs_1 \). Since \( \text{toAscList} \) is applied to both \( xs_1, xs_2 \), then \( xs_1 = xs_2 \).

Therefore, we conclude that for any pair \( X(x_1), X(x_2) \) in \( \hat{t} \),

\[
\pi_1^{-1} \cdot x_1 = \pi_2^{-1} \cdot x_2.
\]

Next we prove that the translation function produces CRS (meta-)terms where the arity of each variable is correctly enforced.

**Property 3.7 (Arity).** Let \( \Delta \vdash t \) be a closed term-in-context and \( \mathcal{T}(\Delta, t) = \hat{t} \) its CRS translation. For each \( X \) that occurs in \( t \), there is a corresponding \( X \) in \( \hat{t} \); moreover, there is some number \( n \) such that all the occurrences of \( X \) in \( \hat{t} \) are in meta-applications of arity \( n \). In other words, in the translated term all the occurrences of \( X \) respect the arity of \( X \).

**Proof.** The translation is syntax directed. For every \( \pi \cdot X \) in \( t \), \( [\pi \cdot X]_\Lambda_t = X(\pi) \) where, by Property 3.6, \( \pi^{-1} \cdot \pi \) is a unique list of variables for all occurrences of \( X(\pi) \) in \( \hat{t} \). This leads to \( \pi \) having the same length for all \( X \) in \( \hat{t} \).

**Property 3.8 (Preservation of Closedness).**

(a) If \( \Delta \vdash t \) is a closed nominal term then its CRS translation \( \hat{t} \) (according to Definition 3.2) is a closed CRS meta-term.

(b) Moreover, if the nominal term \( t \) is ground, then its translation is a CRS term.

**Proof.** This is due to:

1. Our translation respecting the structure of \( t \), which maps atoms to variables, moderated variables to meta-applications consisting of a meta-variable and its corresponding list of arguments in the form \( X^n(a_1, \ldots, a_n) \) (where the arity \( n \geq 0 \) can be
read directly from the meta-term and thus omitted), nominal abstraction to CRS abstraction, nominal functions to CRS functions and nominal tuples to CRS tuples.

(2) As a direct consequence of Properties 3.6 and 3.7, every meta-application \( X^n(\overline{xs}) \)
respects the arity \( n = |\overline{xs}| \) for all occurrences of \( X \) in \( \overline{t} \).

Hence \( \overline{t} \) is a CRS meta-term. If \( \overline{t} \) is ground, \( \overline{t} \) is a CRS term. It remains to prove that \( \overline{t} \) is closed. For this we show that for all occurrences of \( X(\overline{xs}) \) in \( \overline{t} \), \( \overline{xs} \) is a list of bound variables:

By definition of \( T(\Delta, t) \) we know that \( \overline{xs} = \pi \cdot xs \) for some \( \pi \) such that \( \pi \cdot X \) occurs in \( t \). Also, \( xs \equiv \text{toAscList}([\pi^{-1}\cdot \Lambda_t(X)] - \{a \; | \; a\#X \in \Delta\}) \), that is, for each \( a \in xs \) it is a requirement that \( \pi(a) \in \Lambda_t(X) \). If \( \pi(a) \in \Lambda_t(X) \), by Definition 3.1 \( \pi(a) \) occurs abstracted above \( X \). Moreover, as a consequence of Property 3.6 this is the case for all occurrences of \( X \), otherwise \( a\#X \in \Delta \) hence \( \pi(a) \notin \overline{xs} \).

Therefore all the variables in \( \overline{xs} \) are bound, for all occurrences of \( X(\overline{xs}) \) in \( \overline{t} \). Hence we conclude that \( \overline{t} \) is a closed meta-term.

\( \square \)

The following auxiliary lemmas will be useful to prove that \( \alpha \)-equivalent closed nominal terms have a unique CRS translation. Intuitively this is the case since CRS terms are, by definition, considered modulo \( \alpha \).

Lemma 3.9. If \( \Delta \vdash s \approx_{\alpha} t \) then for all \( \pi_s \cdot X \) in \( s \) and \( \pi_t \cdot X \) in \( t \) occurring at position \( p \) in \( s \) and \( t \) respectively, \( \pi^{-1}_s \cdot \overline{xs}_s = \pi^{-1}_t \cdot \overline{xs}_t \), where \( \overline{xs}_s \) and \( \overline{xs}_t \) are the lists computed by the translation function for the translation of the subterms \( \pi_s \cdot X \) in \( s \) and \( \pi_t \cdot X \) in \( t \).

Proof. Direct consequence of the fact that \( \alpha \)-equivalent terms have the same support. \( \square \)

Lemma 3.10. Let \( \Delta \vdash s \) be a nominal term-in-context and \( \Lambda_s \) as defined in Definition 3.1.
Then it is the case that \( \llbracket (a \; b) \cdot s \rrbracket^\Delta_{(a \; b) \cdot \Lambda_s} = \llbracket s \rrbracket^\Delta_{\Lambda_s} \{a \rightarrow b, b \rightarrow a\} \) for all subterms \( s' \) of \( s \).

Proof. By induction on the structure of \( s' \).

- The case \( (s' = c) \). There are three cases to consider. The case where \( c = a \), the case where \( c = b \) and the case where neither \( c \neq a \) and \( c \neq b \).
- Suppose \( c = a \). Then, by Definition 3.2 \( \llbracket b \rrbracket^\Delta_{(a \; b) \cdot \Lambda_s} = b = \llbracket a \rrbracket^\Delta_{(a \rightarrow b, b \rightarrow a)} \).
- Suppose \( c = b \). Then, by Definition 3.2 \( \llbracket a \rrbracket^\Delta_{(a \; b) \cdot \Lambda_s} = a = \llbracket b \rrbracket^\Delta_{(a \rightarrow b, b \rightarrow a)} \).
- Suppose \( c \neq a \) and \( c \neq b \). By Definition 3.2 \( \llbracket c \rrbracket^\Delta_{(a \; b) \cdot \Lambda_s} = c = \llbracket c \rrbracket^\Delta_{\Lambda_s} \{a \rightarrow b, b \rightarrow a\} \).

- The case \( (s' = \pi \cdot X) \). By Definition 3.2 we have \( \llbracket (a \; b) \cdot (\pi \cdot X) \rrbracket^\Delta_{(a \; b) \cdot \Lambda_s} = \llbracket (a \; b) \circ \pi \cdot X \rrbracket^\Delta_{(a \; b) \cdot \Lambda_s} = \llbracket (\pi^{-1} \circ (a \; b)) \cdot c \rrbracket^\Delta_{\Lambda_s} \{a \rightarrow b, b \rightarrow a\} = X(\overline{xs}) \) and also \( \llbracket \pi \cdot X \rrbracket^\Delta_{\Lambda_s} \{a \rightarrow b, b \rightarrow a\} = X(\overline{xs}) \{a \rightarrow b, b \rightarrow a\} \).

For this case we must take into account each atom \( c \) that occurs at a position \( p \) in \( \overline{xs} \) then prove it also occurs at \( p \) in \( \overline{xs}' \{a \rightarrow b, b \rightarrow a\} \). Notice that if \( c \in \overline{xs} \) at \( p \) then, looking at Definition 3.2 \( (\pi^{-1} \circ (a \; b)) \cdot c \in xs \) also at \( p \). Subsequently, \( c \in (a \; b) \cdot \Lambda_s(X) \) and \( (\pi^{-1} \circ (a \; b)) \cdot c \#X \notin \Delta \).
We consider first the case where \( \overline{xs} \neq \emptyset \) then the case where \( \overline{xs} = \emptyset \).

For the case where \( \overline{xs} = \emptyset \), it is also the case that \( (a \; b) \cdot \Lambda_s(X) = \emptyset \). Hence \( \overline{xs}' \{a \rightarrow b, b \rightarrow a\} = \emptyset \) by Definition 3.2.

For the case where \( \overline{xs} \neq \emptyset \), we distinguish on whether \( c = a \), \( c = b \) and finally for any other atom \( c \) such that \( a \neq c \neq b \).
• Suppose $c = a$ such that $a \in \mathcal{X}$ at position $p$. Then $a \in \Lambda(X)$, following Definition 3.2 $\pi^{-1}(b) \in \mathcal{X}$ at position $p$ for both translations, resulting in $b \in \mathcal{X}'$. Hence $a \in \mathcal{X}' \{a \mapsto b, b \mapsto a\}$ at $p$.

• Suppose $c = b$ such that $c \in \mathcal{X}$ at position $p$. Then $b \in \Lambda(X)$, following Definition 3.2 $\pi^{-1}(a) \in \mathcal{X}$ at position $p$ for both translations, resulting in $a \in \mathcal{X}'$. Hence $b \in \mathcal{X}' \{a \mapsto b, b \mapsto a\}$ at $p$.

• Suppose now $c \neq a, c \neq b$ such that $c \in \mathcal{X}$ at position $p$. Then $c \in \Lambda(X)$, thus $c \in \Lambda(X)$. Following Definition 3.2 $\pi^{-1}(c) \in \mathcal{X}$ at position $p$ for both translations, resulting in $c \in \mathcal{X}'$. Hence $c \in \mathcal{X}' \{a \mapsto b, b \mapsto a\}$ at $p$.

The case $(s' = [a]t)$. By Definition 3.2 $[[b](a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} = [b][[(a \cdot b) \cdot t]]_{(a \cdot b) \cdot \Lambda_s}$ where $b \in \Lambda(X)$ occurring in $t$. Take any variable $c$ not occurring free in the CRS term $[[a \cdot t]]_{(a \cdot b) \cdot \Lambda_s}$. Then one could choose a CRS term $[c][[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} \{b \mapsto c\}$ from the class of $\alpha$-equivalent terms. By application of the inductive hypothesis $[c][[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} \{b \mapsto c\} = [c][[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} \{b \mapsto c\} \{b \mapsto a\} \{b \mapsto a\}$. Since variable $c$ does not occur free in $[[a \cdot t]]_{(a \cdot b) \cdot \Lambda_s}$, one could assume $[c][[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} \{b \mapsto c\} \{b \mapsto a\} = [c][[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} \{b \mapsto c\} \{b \mapsto a\}$ without loss of generality. Furthermore, $[[a \cdot t]]_{(a \cdot b) \cdot \Lambda_s} \{b \mapsto a\} = [a][[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} \{b \mapsto a\}]$ where $a \in \Lambda(X)$ for any $X \in \mathcal{X}$ occurring in $t$. Since there exists a variable $c$ not free in $[[a \cdot t]]_{(a \cdot b) \cdot \Lambda_s} \{b \mapsto a\}$ as explained above, we choose an $\alpha$-equivalent CRS term $[c][[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} \{b \mapsto c\} \{b \mapsto a\}$ without loss of generality. And the result follows.

The case $(s = [b]t)$ is similarly solved and thus omitted here.

The case $(s' = [c]t)$. By Definition 3.2 $[[c](a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} = [c][[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s}$ where $c \in \Lambda(X)$ for any $X \in \mathcal{X}$ occurring in $t$.

Furthermore, $[[c]t]_{(a \cdot b) \cdot \Lambda_s} \{a \mapsto b, b \mapsto a\} = [c][[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s} \{a \mapsto b, b \mapsto a\}$ where $c \in \Lambda(X)$ for any $X \in \mathcal{X}$ occurring in $t$.

By inductive hypothesis, $[[a \cdot t]]_{(a \cdot b) \cdot \Lambda_s} = [t]_{(a \cdot b) \cdot \Lambda_s} \{a \mapsto b, b \mapsto a\}$ and the result follows.

The case $(s' = ft)$. By Definition 3.2 $[[f((a \cdot t))]_{(a \cdot b) \cdot \Lambda_s} = f[[[(a \cdot t)]_{(a \cdot b) \cdot \Lambda_s}$. Furthermore, $[[f]t]_{(a \cdot b) \cdot \Lambda_s} \{a \mapsto b, b \mapsto a\} = f[[[t]]_{(a \cdot b) \cdot \Lambda_s} \{a \mapsto b, b \mapsto a\}$. By inductive hypothesis, $[[a \cdot t]]_{(a \cdot b) \cdot \Lambda_s} = [t]_{(a \cdot b) \cdot \Lambda_s} \{a \mapsto b, b \mapsto a\}$ and the result follows.

The case $(s' = (s_1, \ldots, s_n))$. By action of permutation $(a \cdot b)$ on $s'$, $[[a \cdot b](s_1, \ldots, s_n)]_{(a \cdot b) \cdot \Lambda_s} = [[[a \cdot b]s_1, \ldots, (a \cdot b) \cdot s_n)]_{(a \cdot b) \cdot \Lambda_s}$. By Definition 3.2 $[[a \cdot b](s_1, \ldots, (a \cdot b) \cdot s_n)]_{(a \cdot b) \cdot \Lambda_s} = [[[a \cdot b]s_1]_{(a \cdot b) \cdot \Lambda_s}, \ldots, [[[a \cdot b]s_n]_{(a \cdot b) \cdot \Lambda_s}].$

Furthermore, $[[s_1, \ldots, s_n]]_{(a \cdot b) \cdot \Lambda_s} \{a \mapsto b, b \mapsto a\} = \sum_{i=1}^{n} [[[s_i]]_{(a \cdot b) \cdot \Lambda_s} \{a \mapsto b, b \mapsto a\}. By inductive hypothesis, $[[a \cdot b]s_i]_{(a \cdot b) \cdot \Lambda_s} = [s_i]_{(a \cdot b) \cdot \Lambda_s} \{a \mapsto b, b \mapsto a\}$ where $0 \leq i \leq n$ and the result follows.
Theorem 3.11 (Uniformity w.r.t. $\alpha$). Let $\Delta \vdash t$, $\Delta \vdash s$ be a pair of closed nominal terms-in-context such that $\Delta \vdash s \approx_\alpha t$; and let $\hat{s}, \hat{t}$ be their respective CRS translations according to Definition 3.2. Then $\hat{s} = \hat{t}$.

Proof. By induction on the size of $\Delta \vdash s \approx_\alpha t$. We prove $[s']^\Lambda_\hat{\alpha} = [t']^\Lambda_{\hat{\alpha}}$ for all $s'$, $t'$ such that $s' = s|_p$, $t' = t|_p$ and $\Delta \vdash s' \approx_\alpha t'$. $\Lambda_s, \Lambda_t$ are the result of $\Lambda$ on $s, t$ respectively, by Definition 3.1.

We distinguish cases according to the last rule used to derive $\Delta \vdash s' \approx_\alpha t'$.

- The case ($\approx_\alpha a$). Suppose $s' = a$ then, it can only be the case $t' = a$ so that $\Delta \vdash a \approx_\alpha a$. By application of Definition 3.2 $\hat{s} = \hat{a} = \hat{t}$ and the result follows.

- The case ($\approx_\alpha x$). Suppose $ds(\pi_s, \pi_t) \# X \in \Delta$ so that $\Delta \vdash \pi_s.X \approx_\alpha \pi_t.X$. We distinguish cases with respect to $\Lambda_s(X) \cup \Lambda_t(X)$.
  - If $\Lambda_s(X) = \emptyset$ then it can only be the case that $\Lambda_t(X) = \emptyset$ also, by the syntax-directed structure of the derivations in $\approx_\alpha$. Otherwise it contradicts the fact that $\Delta \vdash s \approx_\alpha t$. Therefore, by application of Definition 3.2 $\hat{s} = X = \hat{t}$, and the result follows.
  - If $\Lambda_s(X) \neq \emptyset$, then it is the case $\Lambda_t(X) \neq \emptyset$, as explained in the previous case. Notice that $\Delta$ is shared by both $s$ and $t$. The result follows as consequence of Lemma 3.9.

- The case ($\approx_\alpha [a]$). Suppose $\Delta \vdash s' \approx_\alpha t'$ and $[s']^\Lambda_{\hat{\alpha}} = \hat{s}'$, $[t']^\Lambda_{\hat{\alpha}} = \hat{t}'$ where $a \in \Lambda_s(X)$, $\Lambda_t(X)$ for any variable $X \in V(s) \cup V(t)$. By inductive hypothesis, $\hat{s} = \hat{t}$.
  Using ($\approx_\alpha [a]$), $\Delta \vdash [a]s' \approx_\alpha [a]t'$. Then, $[[a]s']^\Lambda_{\hat{\alpha}} = [a]\hat{s}'$, $[[a]t']^\Lambda_{\hat{\alpha}} = [a]\hat{t}'$ and the result follows.

- The case ($\approx_\alpha [b]$). Suppose $\Delta \vdash (b \cdot a).s' \approx_\alpha t'$, $\Delta \vdash b \# s'$. Let $[(b \cdot a).s']^\Lambda_{\hat{\alpha}} = [t']^\Lambda_{\hat{\alpha}}$ where $a \in \Lambda_s(X)$, $b \in \Lambda_t(X)$ for any variable $X \in V(s) \cup V(t)$. Therefore, by application of Lemma 3.10 $[(b \cdot a).s']^\Lambda_{\hat{\alpha}} = [s']^\Lambda_{\hat{\alpha}} \{a \mapsto b, b \mapsto a\} = \hat{s}' \{a \mapsto b, b \mapsto a\}$. Since $\Delta \vdash b \# s'$ then $\hat{s}' \{a \mapsto b, b \mapsto a\} = \hat{s}' \{a \mapsto b\}$. Also $[t']^\Lambda_{\hat{\alpha}} = \hat{t}'$. By inductive hypothesis, $\hat{s}' \{a \mapsto b\} = \hat{t}'$.
  Using ($\approx_\alpha [b]$), $\Delta \vdash [b]s' \approx_\alpha [b]t'$. Then, $[[b]s']^\Lambda_{\hat{\alpha}} = [b]\hat{s}'$, where $b$ does not occur free in $\hat{s}'$, $[[b]t']^\Lambda_{\hat{\alpha}} = [b]\hat{t}'$ and $[a]s' = [b]s' \{a \mapsto b\}$. The result follows.

- The case ($\approx_\alpha f$). Suppose $\Delta \vdash s' \approx_\alpha t'$ and $[s']^\Lambda_{\hat{\alpha}} = \hat{s}'$, $[t']^\Lambda_{\hat{\alpha}} = \hat{t}'$. By inductive hypothesis, $\hat{s}' = \hat{t}'$.
  Using ($\approx_\alpha f$), $\Delta \vdash fs' \approx_\alpha ft'$. Then, $[[fs']^\Lambda_{\hat{\alpha}} = f\hat{s}'$, $[[ft']^\Lambda_{\hat{\alpha}} = f\hat{t}'$ and the result follows.

- The case ($\approx_\alpha \text{tup}$). Suppose $\Delta \vdash s'_1 \approx_\alpha t'_1$, ..., $\Delta \vdash s'_n \approx_\alpha t'_n$ and $[s'_1]^\Lambda_{\hat{\alpha}} = \hat{s}'_1$, $[t'_1]^\Lambda_{\hat{\alpha}} = \hat{t}'_1$ for $1 \leq i \leq n$. By inductive hypothesis, $\hat{s}'_i = \hat{t}'_i$.
  Using ($\approx_\alpha \text{tup}$), $\Delta \vdash (s'_1, ..., s'_n) \approx_\alpha (t'_1, ..., t'_n)$. Then, $[[s'_1, ..., s'_n]]^\Lambda_{\hat{\alpha}} = (\hat{s}'_1, ..., \hat{s}'_n)$, $[[t'_1, ..., t'_n]]^\Lambda_{\hat{\alpha}} = (\hat{t}'_1, ..., \hat{t}'_n)$ and the result follows.

$\square$
4. Transforming NRS Rules

NRS rules are more general than CRS rules in that free atoms may occur in rules. In this section, we impose some conditions on NRS rules to obtain a class of rules that can be translated to CRS rules.

**Definition 4.1** (Standard Nominal Rule). A nominal rule is called standard when it is closed and the left-hand side has the form $f$s.

**Definition 4.2** (Rule Translation Function). Let $R = \nabla \vdash l \rightarrow r$ be a standard nominal rule.

We define the translation of $R$ as $T^R(\nabla, l, r) = T(\nabla, l) \Rightarrow T(\nabla, r)$ where $T(\Delta, t)$ is given in Definition 3.2.

**Lemma 4.3** (Well-Defined Rule Translation). Let $R = \nabla \vdash l \rightarrow r$ be a standard nominal rule. If $R' = \hat{l} \Rightarrow \hat{r}$ is its translation according to Definition 4.2, then $R'$ is a CRS rule.

**Proof.** First, note that if a nominal rule $\nabla \vdash l \rightarrow r$ is closed (i.e., $\nabla \vdash (l, r)$ is closed), then $\nabla \vdash l$ and $\nabla \vdash r$ are both closed terms. Hence:

- By Property 3.8 both $l$ and $\hat{r}$ are closed CRS meta-terms.
- By definition of a nominal rule, the variables in $r$ are also in $l$. It is easy to see, by induction on Definition 3.2, that $\hat{r}$ contains only those meta-variables occurring in $l$, and meta-variables occur only in meta-applications where the arguments are lists of bound variables respecting the arity of the meta-variable (see Property 3.7). Moreover, $[l]_{\Lambda_l} = [\hat{l}]_{\Lambda(\hat{l}, \hat{r})}$ and $[r]_{\Delta_l} = [\hat{r}]_{\Delta(\hat{l}, \hat{r})}$. This is due to $V(r) \subseteq V(l)$, the property of closedness, and $\nabla$ being shared by all functions.
- By definition of a standard rule (see Definition 4.1), $l$ has the form $f$s.

Hence $R'$ is a CRS rule (see Section 2.2).

**Example 4.4.** The (closed) nominal rules to compute prenex normal forms (see Example 2.7) can be translated to CRS rules by application of our algorithm. We show the CRS translation computed by our Haskell implementation (see 10):

$$\begin{align*}
\text{and}(P, \text{forall}([a]Q(a))) & \Rightarrow \text{forall}([a]\text{and}(P, Q(a))) \\
\text{and}(\text{forall}([a]Q(a)), P) & \Rightarrow \text{forall}([a]\text{and}(Q(a), P)) \\
\text{or}(P, \text{forall}([a]Q(a)), P) & \Rightarrow \text{forall}([a]\text{or}(Q(a), P)) \\
\text{or}(\text{forall}([a]Q(a)), P) & \Rightarrow \text{forall}([a]\text{or}(Q(a), P)) \\
\text{and}(P, \text{exists}([a]Q(a))) & \Rightarrow \text{exists}([a]\text{and}(P, Q(a))) \\
\text{and}(\text{exists}([a]Q(a)), P) & \Rightarrow \text{exists}([a]\text{and}(Q(a), P)) \\
\text{or}(P, \text{exists}([a]Q(a))) & \Rightarrow \text{exists}([a]\text{or}(P, Q(a))) \\
\text{or}(\text{exists}([a]Q(a)), P) & \Rightarrow \text{exists}([a]\text{or}(Q(a), P)) \\
\text{not}(\text{exists}([a]Q(a))) & \Rightarrow \text{forall}([a]\text{not}(Q(a))) \\
\text{not}(\text{forall}([a]Q(a))) & \Rightarrow \text{exists}([a]\text{not}(Q(a)))
\end{align*}$$

Note that the nominal variable $P$ becomes the CRS meta-variable $P$ of arity 0. Hence, by definition (see 21), if a substitute of $P$ contains the free variable $a$, then the bound variable $a$ in the meta-term will be renamed to avoid name clashes. On the other hand, the nominal variable $Q$ becomes the CRS meta-variable $Q$ of arity 1, which has the bound variable $a$ as argument.
Example 4.5. The next set of nominal rules are inspired by the simulation of $\beta$-reduction and $\eta$-reduction as defined in [13].

\[
\begin{align*}
(\beta_{\text{app}}) & \quad \text{app}(\text{lam}([a]\text{app}(X,X'), Y)) \Rightarrow \text{app}(\text{app}(\text{lam}([a]X'), Y), \text{app}(\text{lam}([a]X), Y))) \\
(\beta_{\text{var}}) & \quad \text{app}(\text{lam}([a]X), X') \Rightarrow X \\
(\beta_{\lambda}) & \quad a\#Y \vdash \text{app}(\text{lam}([a]Y), X) \Rightarrow Y \\
(\beta_{\text{lam}}) & \quad b\#Y \vdash \text{app}(\text{lam}([a]\text{lam}([b]X)), Y) \Rightarrow \text{lam}([b]\text{app}(\text{lam}([a]X), Y)) \\
(\eta) & \quad a\#X \vdash \text{lam}([a]\text{app}(X,a)) \Rightarrow X
\end{align*}
\]

The CRS translation is:

\[
\begin{align*}
(\beta_{\text{app}}) & \quad \text{app}(\text{lam}([a]\text{app}(X(a), X'(a))), Y) \Rightarrow \text{app}(\text{app}(\text{lam}([a]X'(a)), Y), \text{app}(\text{lam}([a]X(a)), Y))) \\
(\beta_{\text{var}}) & \quad \text{app}(\text{lam}([a]X), X') \Rightarrow X \\
(\beta_{\lambda}) & \quad \text{app}(\text{lam}([a]Y), X) \Rightarrow Y \\
(\beta_{\text{lam}}) & \quad \text{app}(\text{lam}([a]\text{lam}([b]X(a,b))), Y) \Rightarrow \text{lam}([b]\text{app}(\text{lam}([a]X(a,b)), Y)) \\
(\eta) & \quad \text{lam}([a]\text{app}(X,a)) \Rightarrow X
\end{align*}
\]

In rule ($\beta_{\text{lam}}$), notice how both occurrences of the meta-variable $X$ share the same ordered list of bound variables, regardless of the fact that on the left-hand side, $[a]$ is above $[b]$ on the syntax tree while on the right-hand side is the opposite. This ensures that substitutions work well, as explained in more detail in the next section.

5. Simulating Nominal Rewrite Steps

We consider next the relationship between the rewriting relation on nominal terms generated by a set of nominal rules $R$ and the rewriting relation on CRS terms generated by its translation. Our goal is to show that the rewriting relation is preserved when nominal terms and rules are translated to CRSs.

Translation of a rewrite relation is not as straight-forward as one could expect. The rewriting relation generated by a set of CRS rules is defined on terms, not on meta-terms. Recall that CRS substitutes are terms, containing no meta-variables, preceded by the binder $\lambda$ and a list of pairwise distinct variables (the length of the list corresponds with the arity of the meta-variable it substitutes). In order to preserve the rewriting relation, we need to consider only ground nominal substitutions. Moreover, substitutions will not be translated on their own, but together with the term-in-context to be instantiated (since permutations are also applied to the substitution in order to preserve the meaning of the term). For this reason, we will define a translation function for pairs of a term-in-context $\Delta \vdash t$ and a substitution $\sigma$.

Moreover, there are swappings occurring in a permutation that can only be dealt with by applying directly the permutation to the nominal substitution before translation. These swappings correspond to mappings from atoms to unabstracted atoms occurring in the term. Dealing with these swappings at term level would contradict the property of closedness of a CRS rule. Take for instance the example

\[
(\Delta \vdash (a \cdot b) \cdot X, [X \mapsto f(a,b)])
\]

The term $\vdash (a \cdot b) \cdot X$ is trivially closed (no unabstracted atoms occur in the term and there is only one variable). The CRS translation given in Definition 3.2 for nominal terms and
Definition 5.1 for substitution, given below, produce the pair
\[(X, [X \mapsto f(b, a)])\]
where the permutation \((a \ b)\) has been directly applied to the instantiation of \(X, f(b, a)\) to construct the CRS substitute and not in the nominal term \(X\), since neither \(a\) nor \(b\) are under abstractions above \(X\). Accordingly, we must also take into account alterations to the binding structure by direct application of the permutation to the instantiating term, that is, the list of bindings added to the translation of \(\sigma\) (possibly empty, as the above example shows) must be modified to preserve the scoping after application of \(\pi\) to \(\sigma\). This is safe since, by definition of closed terms (see Definition 2.9), if for any two occurrences \(\pi_i.X, \pi_j.X\) in a term \(t\) such that \(a \in \text{support}(\pi_i)\) and \(\pi_i(a) \notin \Lambda_t(X)\), it is also the case that \(\pi_j(a) \notin \Lambda_t(X)\) to preserve closedness, hence \(a \in \text{support}(\pi_j)\). Moreover, it has to be that \(\pi_i(a) = \pi_j(a)\) otherwise \(a \# X \in \Delta\). Further examples are considered after Definition 5.1 where we present the nominal substitution translation function.

We must ensure a nominal substitution is correctly translated with respect to the nominal term it instantiates. For this, we apply the function \(\text{toAscList}\) equally in both Definition 3.2 and Definition 5.1 over the set of mappings \(\Lambda\) (see Definition 3.1), which produces a fixed and ordered list of atoms \([a_1, \ldots, a_n]\) for each nominal variable in the term. These lists are added to the substitutes for meta-variables, which have the form \(\Delta(a_1, \ldots, a_n)t\).

**Definition 5.1 (Substitution Translation).** Let \(\Delta \vdash t\) be a closed nominal term-in-context, \(\Lambda_t\) as in Definition 3.2 and \(\sigma\) a nominal substitution satisfying \(\Delta\), such that \(\sigma = [X_i \mapsto t_i], 1 \leq i \leq n\) where \(\text{dom}(\sigma) \subseteq V(t)\) and \(\sigma\) is ground.

Then \(\mathcal{T}^f(\Delta, t, \sigma) = [X_i \mapsto \Delta(\pi s_i).s_i]\) is defined as follows:
- \(\pi s_i \triangleq \pi_i \cdot xs_i\) and,
- \(xs_i \triangleq \text{toAscList}([\pi_i^{-1}.\Lambda_t(X_i)] - \{a \mid a \# X_i \in \Delta\})\),
- \(s_i \triangleq \mathcal{T}(\Delta, \pi_i \cdot t_i)\) where \(\pi_i\) is the permutation suspended in the leftmost occurrence of \(X_i\) in \(t\).

Lemma 5.4 justifies the use of the leftmost occurrence of \(\pi \cdot X\) in \(t\). Intuitively, each substitute generated by application of the translation function to distinct occurrences of a moderated variable is indeed \(\alpha\)-equivalent. Hence the leftmost occurrence is used as a representative.

We denote by \((\hat{t}, \hat{\sigma})\) the result of \((\mathcal{T}(\Delta, t), \mathcal{T}^f(\Delta, t, \sigma))\).

**Example 5.2.** Consider the following pair of a nominal term-in-context and substitution
\(t = \vdash f([a][b] X, [b][a] X), \sigma = [X \mapsto g(a, b)]\)
where
\(t\sigma = f([a][b]g(a, b), [b][a]g(a, b))\).

Then, applying \((\mathcal{T}, \mathcal{T}^f)\) we obtain the pair
\(\hat{t} = f([a][b]X(a, b), [b][a]X(a, b)), \hat{\sigma} = [X \mapsto \Delta(a,b), g(a, b)]\).

The CRS term \(\hat{t}\sigma\) is computed as follows:
\[f([a][b](\Delta(a,b).g(a, b))(a, b), [b][a](\Delta(a,b).g(a, b))(a, b)) \rightarrow_\beta\]
\[f([a][b]g(a, b), [b][a]g(a, b))\]
which corresponds to the nominal term \( t\sigma \).

**Example 5.3.** We revisit the nominal term

\[
\vdash [a][b](a\;c)\cdot X
\]

given in example 3.4 for a more detailed view on its translation to CRSs.

Let’s assume we are given the pair

\[
(\vdash [a][b](a\;c)\cdot X, \sigma = [X\mapsto f(a,b,c)] ).
\]

Its CRS translation is

\[
( [a][b]X(b,a), \hat{\sigma} = [X\mapsto \lambda(b,a).f(c,b,a)] )
\]

with \( xs = [b,c], \bar{xs} = [b,a] \) and \((a\;c)\cdot X = f(c,b,a)\). Notice we must apply \((a\;c)\) to \(xs\) in order to allow capture of the variable \(a\) occurring in \((a\;c)\cdot X\).

It should now be clear that the meta-application in the translated term

\[
[a][b]X(b,a)
\]

does not swap the variables \(a, b\) but maps \(b \mapsto b, c \mapsto a\) instead.

Later we formally prove the application of permutations to the nominal substitution \(\sigma\) does not affect the uniqueness of \(\hat{\sigma}\) modulo \(\alpha\), when more than one occurrence of the same meta-variable appears in the term. Intuitively, if there are more occurrences of \(X\), Property 3.8 implies the initial list of bindings we call \(xs\) is syntactically equivalent for all occurrences of \(X\) thus it binds the same atoms. Considering \(\pi\) is applied to both \(xs\) and nominal substitution \(\sigma(X)\), renamings do not affect the structure of the binding. Moreover, for any other renaming of atoms \(a \mapsto \pi(a)\) occurring during application of \(\pi\cdot\sigma(X)\), an identical mapping must exist in all other occurrences of \(\pi\cdot X\) else it contradicts the property of closedness (see Definition 2.9) therefore \(a\#X \notin \Delta\).

For instance, consider the following pair of a closed term-in-context and a substitution.

\[
(a, c\#X \vdash g([a][b][c](a\;d)(e\;f)\cdot X), [a][b][c](c\;d)(e\;f)\cdot X, [X\mapsto f(b,d,e,f)]).
\]

The term translation function produces a CRS meta-term

\[
g([a][b][c]X(b,a), [a][b][c]X(b,c))
\]

and the substitution translation produces the corresponding substitute

\[
[X\mapsto \lambda(b,a).f(b,a,f,e)].
\]

Note that, if translation of the substitution is done with respect to each variable in the term, we obtain:

\[
[X\mapsto \lambda(b,a).f(b,a,f,e)] \text{ and } [X\mapsto \lambda(b,c).f(b,c,f,e)]
\]

for each occurrence of \(X\). These substitutes are indeed \(\alpha\)-equivalent and our algorithm outputs the leftmost \( [X\mapsto \lambda(b,a).f(b,a,f,e)] \).

Moreover, applying the lexical ordering directly to \(\bar{xs}\) instead of \(xs\) would produce substitutes which are no longer \(\alpha\)-equivalent, providing incorrect instantiations.

We now formalize the property of equivalence modulo \(\alpha\) of substitution occurrences after translation. The intuition is they are equivalent because of terms being closed (see Definition 2.9), sharing an initially equivalent variable binding list known as \(xs\) (see Property 3.6). It is this closedness of the term that preserves \(\alpha\)-equivalence of substitutes after application of permutations to both nominal substitution and list \(xs\).
Lemma 5.4 (α-equivalence of Substitutes). Let $\Delta \vdash t$ be a closed nominal term-in-context, $\Lambda_t$ as defined in Definition 3.1, and $\sigma$ a nominal substitution satisfying $\Delta$ such that $\text{dom}(\sigma) \subseteq V(t)$ and $t\sigma$ is ground. Let $\pi_i : X, \pi_j : X$ be two occurrences of the same variable in $t$, and let $[X \mapsto \Lambda(\pi_i).s_i]$ and $[X \mapsto \Lambda(\pi_j).s_j]$ be translations according to Definition 5.7 but using $\pi_i$ and $\pi_j$ respectively. Then $[X \mapsto \Lambda(\pi_i).s_i] \approx_\alpha [X \mapsto \Lambda(\pi_j).s_j]$.

Proof. By definition we know that $T^I(\Delta, t, [X_i \mapsto t_i]) = [X_i \mapsto \Lambda(\pi_i).s_i]$ with $\pi_i \triangleq \pi_i.xs_i$, $xs_i \triangleq \text{toList}([\pi_i^{-1}.\Lambda_t(X_i)] - \{a | a \# X_i \in \Delta\})$ and $s_i \triangleq T(\Delta, \pi_i, t_i)$, if $\pi_i : X_i$ is the leftmost occurrence of $X_i$.

Hence, each atom $a \in \text{support}(\pi_i)$ with $\pi_i(a) \not\in \Lambda_t(X)$ must satisfy $\pi_i(a) = \pi_j(a)$, so that when $\pi_i, \pi_j$ are applied to each occurrence of $t_i$ during translation, they remain equivalent, therefore $s_i = s_j$. Otherwise $a \in \text{ds}(\pi_i, \pi_j)$ such that $a \# X \in \Delta$ by Definition 2.9. Since $\sigma$ must also satisfy $\Delta$, it is the case that $a \# t_i$, and application of either $\pi_i$ or $\pi_j$ to $t_i$ produces no changes.

Now we look at the binding list added to the substitute (i.e., $\pi_i.xs_i, \pi_j.xs_j$). Property 3.6 states that $xs$ is shared by all occurrences, and the term is closed, then it is the case that each variable in $xs$ binds the same variable in $t_i$ for both occurrences. It immediately follows that by application of $\pi_i, \pi_j$ to each occurrence of $xs$ and $t_i$, the renaming of bound variables does not affect the binding structure, hence they are α-equivalent. Finally, for any other atom $a \in \Lambda_t(X)$ but $a \not\in xs$ it can only be that $\pi^{-1}(a) \# X \in \Delta$ hence $\pi^{-1}(a) \not\in xs$. It also does not occur free in $t_i$ since it must satisfy $\Delta$, as previously stated. Therefore it does not alter the outcome of the translation.

This shows that the choice of the leftmost element in the translation does not affect correctness.

We are now ready to prove that substitutions are correctly translated.

Lemma 5.5 (Instantiation). Let $\Delta \vdash t$ be a closed nominal term-in-context, $\Lambda_t$ as defined in Definition 3.1, and $\sigma$ a substitution satisfying $\Delta$ such that $\text{dom}(\sigma) \subseteq V(t)$ and $t\sigma$ is ground.

Assume $[[t]]^\Delta_{\Lambda_t}, T^I(\Delta, t, \sigma) = (t', \hat{\sigma})$, where $t'$ is any subterm of $t$ (e.g. $t' = t$).

Then $[[t']\sigma]^\Delta_{\Lambda_t} = (t'\hat{\sigma})$.

Proof. By induction on the structure of $t'$.

(1) If $\Lambda_t(X) = \emptyset$,
then it immediately follows that $[[\pi \cdot X]]^\Delta_{\hat{\sigma}} = (X, \hat{\sigma})$ where $\hat{\sigma}(X) = s \triangleq T(\Delta, \pi\sigma(X))$ by definition of substitution translation. Therefore $X\hat{\sigma} = s$. This is equivalent to $[[\pi \cdot X]\sigma(X)]_{\hat{\sigma}} = s$.

(2) If $\Lambda_t(X) \neq \emptyset$,
then $[[\pi \cdot X]]^\Delta_{\hat{\sigma}} = (X(\pi s), \hat{\sigma}(X) = \Lambda(\pi s).s), where s = T(\Delta, \pi\sigma(X))$. Therefore $(X(\pi s))\hat{\sigma}(X) = (\Lambda(\pi s), s(\pi s)) = s$. This is also equivalent to $[[\pi \cdot X]\sigma(X)]_{\hat{\sigma}} = s$.

(3) If $t' = [a]s$ then
$[[a]s]\sigma]^\Delta_{\Lambda_t} = [[a]s\sigma]^\Delta_{\Lambda_t} = [a]s\hat{\sigma}$ where $\hat{\sigma} = [s\sigma]^\Delta_{\Lambda_t}$ by IH.
(fun) If $t' = fs$ then
\[ \langle [fs]s \rangle_{\Lambda_t}^\Delta = \langle [f]s \rangle^\Delta \] where $\check{s} \check{\sigma} = [s]s_{\Lambda_t}$ by IH.

(tuple) If $t' = (s_1, \ldots, s_n)$ then
\[ \langle [s_1, \ldots, s_n]s \rangle_{\Lambda_t}^\Delta = \langle [s_1s_1]_{V(s_1)}, \ldots, [s_n]s_{V(s_n)} \rangle_{\Lambda_t}^\Delta = (\check{s}_1 \check{\sigma}_1, \ldots, \check{s}_n \check{\sigma}_n) \] where each $\check{s}_i \check{\sigma}_i = [s_i]s_{V(s_i)}_{\Lambda_t}$ by IH.

And $\check{s}_i \check{\sigma}_i|_{V(s_i)} = \check{s}_i \check{\sigma}_i$, as a consequence of Lemma 5.4.

Nominal variable translation depends both on freshness context and abstractions occurring above the variable. The translation function uses them to build both the list of arguments of a meta-application and the list of binders added to a substitute, whereas the syntax-directed nature of the translating function transforms the rest of the elements directly. By keeping track of the abstractions above a variable via the function $\Lambda$, translating any subterm $t'$ of $t$ with $\Lambda_t(X)$ results in the same term as translating $t'$ within $t$, as stated in the following lemma.

Note that $C[]$ is a term, as explained in the paragraph above Definition 2.8 and is translated to $\bar{C}[]$ using Definition 3.2.

**Lemma 5.6** (Context). Let $\Delta \vdash t$ be a nominal term-in-context such that $t = C[s]$ (i.e., $t = C[\rho \mapsto s]$), where $s$ is a ground nominal term.
Assume $\langle C \rangle^\Delta_{\Lambda_C} = \bar{C}$ and $\langle s \rangle^\Delta = \check{s}$.
Then $\langle C[s] \rangle^\Delta_{\Lambda_t} = \bar{C}[s]$.

**Proof.** This is a particular case of Lemma 5.5 and $\sigma = [\rho \mapsto s]$ where if $T^f(\Delta, t, \sigma) = \hat{\sigma}$ then $\langle C[s] \rangle^\Delta_{\Lambda_t} = \bar{C}[s]$ since $s$ is ground, hence $\Lambda_C = \Lambda_t$.

We can now derive the main result of the paper: the preservation of the rewrite relation under the translation.

**Theorem 5.7** (Rewrite Step Translation). Let $R = \nabla \vdash l \rightarrow r$ be a standard nominal rule. Let $t$ be a ground nominal term, $t = T(\varnothing, t)$.
If $t \rightarrow_R u$ then, $i \Rightarrow R' \check{u}$ using $R' = T^R(\nabla, l, r)$, and $\check{u} = T(\varnothing, u)$.

**Proof.** If $t \rightarrow_R u$ then there exists $C, \sigma$ such that $t \approx \alpha C[l\sigma]$ with $\sigma$ a ground nominal substitution satisfying $\nabla$ such that $\text{dom}(\sigma) \subseteq \text{V}(l)$.

Also $R' = T^R(\nabla, l, r) = [l]_{\Lambda}^\nabla \Rightarrow [r]_{\Lambda}^\nabla = \check{i} \Rightarrow \check{r}$ by Definition 4.2, where Lemma 4.3 asserts that the translation is a CRS rule.
If we have, by application of Definition 5.1, $T^f(\nabla, l, \sigma) = \hat{\sigma}_l$ then, by Lemma 5.5, $[l\sigma]_{\Lambda_{ls}}^\nabla = \check{l\sigma}_l$.
Hence we have $\check{i} = \bar{C}[\check{\sigma}_l]$ by Lemmas 5.6 and Property 3.11.

Similarly, since $u \approx \alpha C[\sigma_r]$ we have $T^f(\nabla, r, \sigma) = \hat{\sigma}_r$, leading to $\check{u} = \bar{C}[\check{\sigma}_r]$ by application of Definition 5.1, followed by Lemmas 5.5, 5.6 and Property 3.11. Notice that $\text{dom}(\check{\sigma}_r) \subseteq \text{dom}(\check{\sigma}_l)$ and $\check{\sigma}_r(\check{X}) \approx \alpha \check{\sigma}_l(\check{X})$ by Lemma 5.4.

Hence we conclude by stating that if $l\sigma \rightarrow_R r\sigma$ then $i \Rightarrow R' \check{u}$ as expected.

**Corollary 5.8** (Termination). Termination of the translated CRS implies termination of the NRS.
6. **Transforming from CRSs to NRSs: An Improved Approach**

Being our ultimate goal to obtain a tool capable of transforming rules back and forth between CRSs and NRSs as necessary, we consider as a requisite an example that shows the full power of applying both translating functions. To that extent, and based on the CRS to NRS translation defined in \[14\], we offer an improved algorithm to translate CRS rules to closed nominal rules.

Since CRS rules follow Barendregt’s naming convention (each abstraction uses a different bound variable in order to avoid name clashes), we will work with closed rewriting (see Definition 2.10), where in each rewriting step a freshened copy of the rule is chosen (that is, the copy is freshened with respect to itself and the term to be rewritten). Closed rewriting makes the translation simple, delivering a behaviour closer to CRSs.

### 6.1. Translating Meta-terms in CRS Rewrite Rules

We begin by defining a pair of auxiliary functions.

Function $\phi$ contains the leftmost meta-application for each meta-variable occurring in the left-hand side of a CRS rule $l$, that is, a meta-variable $Z^n_i$ and its respective leftmost variable argument list $(a_1, \ldots, a_n)$ such that $\phi(Z^n_i) = [a_1, \ldots, a_n]$. Then, each $\phi(Z^n_i)$ in $l$ provides a concrete representative for the $\alpha$-equivalence class of $l$ since $\phi(Z^n_i)$ contains all the variables bound above $Z^n_i$. This is instrumental for the preservation of both closedness (see Lemmas \[6.7\] & \[6.8\]) and the rewriting relation (see Theorem \[6.16\]) of CRS rules when translated to NRSs.

The notion behind the second auxiliary function, $\Psi$, was briefly introduced in Section 2.3. We aim to convert each meta-application of form $Z^n_i(a_1, \ldots, a_n)$ occurring in a left CRS rule $l$ into a list of swappings suspended on a NRS variable $\pi_i \cdot Z^n_i$ which, when instantiated, simulates the $\beta$-reduction of a valuation $\sigma$ applied to each occurrence of $Z^n_i$. To accomplish this, $\Psi$ is parameterised by the concrete representative of each meta-variable, $\phi_l(Z^n_i)$, and applied locally to each argument list $(b_1, \ldots, b_n)$ belonging to a meta-application of $Z^n_i$ which is not the leftmost one, in order to preserve $\alpha$-equality along the NRS translation.

Hence, both auxiliary functions will be of use when defining the translation algorithm for left- and right-hand sides of CRS rules (Definitions \[6.3\] & \[6.6\]). We start by providing a formal definition of $\phi$.

**Definition 6.1.** Given a closed CRS meta-term $t$, the partial mapping $\Phi_t$ from meta-variables to lists of variables is defined such that

$$\Phi_t(Z^n_i) = [a_1, \ldots, a_n]$$

if the leftmost occurrence of the meta-variable $Z^n_i$ in $t$ has the form $Z^n_i(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n$ are pairwise distinct bound variables. We denote by $\Phi_t(Z^n_i)_k$ the $k$th element in the list $\Phi_t(Z^n_i)$.

To provide a behaviour similar to that of CRS, NRSs must maintain the relation among argument lists occurring for a meta-variable $Z^n_i$ along a CRS meta-term. This relation is one of position within the argument list such that all (possibly distinct) variables at a position $k$, $1 \leq k \leq n$, for each argument list adjacent to an occurrence of $Z^n_i$, are $\beta$-reduced by application of binder $\lambda x_k$ in a substitute for a valuation $\sigma(Z^n_i)$. Furthermore, consider a *non left-linear* closed meta-term (i.e., more than one occurrence of a meta-variable exists) of form $([a]Z^n_i(a), [b]Z^n_i(b))$. By the property of $\alpha$-equivalence in CRSs, abstracted variables
\(a, b\) are considered syntactically equal. This is not the case for NRSs, where \(\alpha\)-equivalence is logical, deeming the direct translation to NRSs \([(a)Z, (b)Z]\) unsuitable since a closed meta-term is translated to an non-closed NRS term. Both translation issues are approached by the same methodology: to make use of the NRS tools to check \(\alpha\)-equivalence of terms, that is, swappings and freshness constraints.

To construct a list of swappings for an occurrence of a non-leftmost NRS variable \(Z\) with respect to both \(\phi_t(Z)^n\) and a variable argument list \((b_1, \ldots, b_n)\) adjacent to that same CRS occurrence \(Z^n\), is to convert a permutation in two-line notation into a series of \(k\)-cycles where \(k \geq 2\) (\(\text{Id}\) permutations are discarded) followed by a decomposition into swappings, as explained in Example 2.18. Notice that the set of variables in \(\phi_t(Z)^n\) and \((b_1, \ldots, b_n)\) can be equivalent, disjoint or sharing some variables, therefore it is not a permutation per se. However, variables are distinct among them on both lists and length of the list is shared by both such that there is a one-to-one correspondence between each variable in \(\phi_t(Z)^n\) and \((b_1, \ldots, b_n)\). This is enough to define a cycle notation (therefore there would be some abuse of notation when using the word permutation for the translation of CRS meta-applications).

Take for instance the closed CRS meta-term

\[
t = [a][b][c]Z(a, b, c), [x][c][y]Z(x, c, y)
\]

where variable name \(c\) is shared between meta-applications. Application of \(\phi\) to \(t\) generates \(\phi_t(Z) = [a, b, c]\) and, to convert the set of mappings \([a \rightarrow x, b \rightarrow c, c \rightarrow y]\) from the leftmost argument list to the rightmost one into a list of swappings, we transform the implicit two-line notation (variables in the domain of the mapping on the top row and their respective image on the bottom row) into the cycle notation \((a, x)(b, c, y)\) and finally into the list of swappings \(\pi_Z = (y b)(c b)(x a)\) as informally described in Section 2.3. Then, \(\pi_Z(a) = x, \pi_Z(b) = c, \pi_Z(c) = y\) as expected. Also, \(\pi_Z(x) = a\) and \(\pi_Z(y) = c\), however both \(x, y\) do not belong to the domain \(\phi_t(Z)\) of the mappings therefore their image is irrelevant for a correct translation of the permutation list. Moreover, it will be shown in the translation Definition 6.3 that both atoms must be fresh for the nominal variable \(Z\), i.e., \(x \notin Z, y \notin Z \in \Delta\) to preserve closedness of the NRS translation. Hence one could say that auxiliary function \(\Psi\) upgrades directed paths (i.e., finite sequence of arcs connecting distinct vertices without repetition) to cycles. Next is the formal definition of \(\Psi\).

When constructing cycles, the main impediment is that if the initial element chosen by \(\Psi\) is part of a directed path then, there may be elements preceding it in the sequence which are not reachable by constructing the sequence rightwards from the initial element. The solution, to construct the sequence leftwards from the initial element only in the case \(\Psi\) is not dealing with a real cycle (thus it is a directed path). Here is how.

**Definition 6.2.** Let \(s = [a_1, \ldots, a_n], t = [b_1, \ldots, b_n]\) be any two pairs of lists of length \(n\) over the set of CRS variables \(\mathcal{V}\), and \(f : \mathcal{V}_s \rightarrow \mathcal{V}_t, f^{-1} : \mathcal{V}_t \rightarrow \mathcal{V}_s\) a pair of mappings such that \(b_k = f(a_k)\) and \(a_k = f^{-1}(b_k)\), for \(1 \leq k \leq n\). Then, \(\pi = \Psi(s, t)\) where \(\pi\) is a list of swappings over the set of atoms \(\mathcal{A}\) and \(\Psi\) is a function recursively defined as follows:

\[
\Psi([\text{nil}, \text{nil}]) = \text{Id}
\]

\[
\Psi([a_1, \ldots, a_n], [b_1, \ldots, b_n]) = (a_m b_k)(a_m b_j) \cdots (a_m b_1)(a_m a_1)(a_m a_i) \cdots (a_m a_t) \circ \Psi(s_1, t_1) \quad (\text{where } 1 \leq i, j, k, l, m \leq n) \quad \text{and}
\]

\[
\bullet (a_m b_k)(a_m b_j) \cdots (a_m b_1)(a_m a_1)(a_m a_i) \cdots (a_m a_t) \quad \text{is the 2-cycle decomposition by Definition 2.17 of the permutation in cycle form } C = (a_m, a_t, \ldots, a_i, a_1, b_1, \ldots, b_j, b_k);
\]

\[
\text{Definition 6.3.} \quad \text{Let } \mathcal{V} = \{Z_1, \ldots, Z_n\}; \text{ then, }
\]

\[
\Psi([a_1, \ldots, a_n], [b_1, \ldots, b_n]) = (a_m b_k)(a_m b_j) \cdots (a_m b_1)(a_m a_1)(a_m a_i) \cdots (a_m a_t) \circ \Psi(s_1, t_1) \quad (\text{where } 1 \leq i, j, k, l, m \leq n) \quad \text{and}
\]

\[
\bullet (a_m b_k)(a_m b_j) \cdots (a_m b_1)(a_m a_1)(a_m a_i) \cdots (a_m a_t) \quad \text{is the 2-cycle decomposition by Definition 2.17 of the permutation in cycle form } C = (a_m, a_t, \ldots, a_i, a_1, b_1, \ldots, b_j, b_k);
\]
• $C$ is constructed by recursive application of functions $f$ and $f^{-1}$ over $a_1$ as follows:

$$a_m \xleftrightharpoons f^{-1}(a_i) \xleftrightharpoons a_i \xleftrightharpoons f^{-1}(a_i) \xleftrightharpoons a_i \xleftrightharpoons f^{-1}(a_i) \xleftrightharpoons a_i \xleftrightharpoons f(a_1) \xleftrightharpoons b_1 \xleftrightharpoons \cdots \xleftrightharpoons b_j \xleftrightharpoons f(b_j) \xleftrightharpoons b_k$$

where $f^{-1}(a_1)$ and $f(b_j)$ are only applicable when $b_k \neq a_1$. Otherwise, if $b_k = a_1$ then the cycle form would be $(a_1, b_1, \ldots, b_j)$, generating then a list of swappings $(b_j, a_1) \cdots (b_1, a_1)$.

• $s_1 = s \setminus C$

• $t_1 = t \setminus C$

Using the definition on the ordered pair of variable lists $[a, b, c], [x, c, y]$ from the above example results in mappings:

$$f(a) = x \quad f(b) = c \quad f(c) = y \quad f^{-1}(x) = a \quad f^{-1}(c) = b \quad f^{-1}(y) = c$$

Then, function $\Psi$ begins by applying $f$ to variable $a$ such that

$$a \xrightarrow{f(a)} x$$

and since $x \not\in \text{domain } f$, $a \not\in \text{domain } f^{-1}$ it returns the cycle $(a, x)$ such that

$$\Psi([a, b, c], [x, c, y]) = (a \ x) \circ \Psi([b, c], [c, y])$$

where lists $[b, c], [c, y]$ are induced by elimination of variables $a, x$ from both lists. Finally,

$$\Psi([b, c], [c, y]) = (b \ y)(b \ c) \circ \text{Id}$$

is created as follows. Starting with variable $b$,

$$b \xrightarrow{f(b)} c \xrightarrow{f(c)} y$$

with $y \not\in \text{domain } f$ and $b \not\in \text{domain } f^{-1}$ and cycle notation $(b, c, y)$. Also, $\Psi(nil, nil) = \text{Id}$.

Notice the positioning of swapping $(a, x)$ is different from the previous example solution. Previously we had $\pi_Z = (y \ b)(c \ b)(x \ a)$ whereas by application of Definition 6.2 it resolves to $(x \ a)(y \ b)(c \ b)$ ($\text{Id}$ is omitted). However, Lemma 2.15 stated that disjoint cycles commute therefore both permutations perform equivalent actions.

To translate a rule $l \Rightarrow r$, two different functions are applied to $l$ and $r$, both parameterised by $\Phi$. We start by looking at the functions to translate CRS meta-terms separately.

The first translation, denoted $C_l(\cdot)$ uses an auxiliary function working on pairs: $(\Delta, l)^{\Lambda}_{\Phi_l}$, such that $C_l(l) = (\varnothing, l)^{\varnothing}_{\Phi_l}$ where $l$ is a meta-term and $\Delta, \Lambda$ are recursively constructed. $\Delta$ has freshness constraints to avoid certain names appearing, in order to keep the nominal term consistently named throughout, since there are no naming conventions in NRSs. $\Lambda$ is a set of variables such that the recursive call $(\Delta, Z^n_i(a_1, \ldots, a_n))^\Lambda_{\Phi_l}$ has in $\Lambda$ those variables bound above $Z^n_i$.

Our interest on $\Lambda$ is, firstly, towards those variables which appear bound above $Z^n_i$ but not as part of its meta-application $Z^n_i(a_1, \ldots, a_n)$, for each occurrence of $Z^n_i$ in $l$, particularly the leftmost one. A freshness condition must be added for each atom $a \in \Lambda$ but not in $Z^n_i(a_1, \ldots, a_n)$ to preserve closedness in the translated NRS rule.

For any other occurrence of a meta-variable $Z^n_i$ that is not the leftmost, $\Lambda \setminus \phi_l(Z^n_i)$ contains the set of atoms that must be fresh for $Z_i$ in the NRS translation to translate consistently from CRSSs to NRSs and back again. That is, $\Lambda \setminus \phi_l(Z^n_i)$ is the set of all variables bound above $Z^n_i$ that cannot be occurring in the translated NRS term, otherwise the NRS term is no longer closed by means of Definition 2.9.
Therefore, the left-hand side translation function does the following: for each meta-variable \( Z^n_i \) in \( l \), if \( Z^n_i \) is the leftmost subterm of the form \( Z^n_i(a_1, \ldots, a_n) \), it is replaced by \( Z_i \) and for each \( a \in \Lambda \) but not in \( Z^n_i(a_1, \ldots, a_n) \), \( a \# Z_i \) is added to \( \Delta \), whereas the rest of the subterms with the form \( Z^n_i(b_1, \ldots, b_n) \) are replaced by \( \Psi(\phi_l(Z^n_i), (b_1, \ldots, b_n)) \cdot Z_i \). Additionally, \( b_j \# Z_i \) is added to \( \Delta \) for each \( b_j \in \Lambda \setminus \phi_l(Z^n_i), 1 \leq j \leq n \), to preserve closedness of the translation. No further freshness constraints are needed since we are working with closed nominal rewriting.

We provide examples after the formal definition.

**Definition 6.3** (Left Translation). Let \( s \) be a closed CRS meta-term, \( \Phi \) the function defined in Definition 6.1 and \( \Psi \) the function in Definition 6.2. Then \( C_l(s) = (\emptyset, s)_{\Phi, s} \), where \((\Delta, s)_{\Phi, s}^\Lambda\) is inductively defined as follows:

\[
\begin{align*}
(\Delta, a)_{\Phi, s}^\Lambda & = (\Delta, a) \\
(\Delta, [a]t)_{\Phi, s}^\Lambda & = (\Delta', [a]t'), \\
& \quad \text{where } (\Delta', t') = (\Delta, t)_{\Phi, s}^{\Lambda \cup \{a\}} \\
(\Delta, f(t_1, \ldots, t_n))_{\Phi, s}^\Lambda & = (\Delta', f(t'_1, \ldots, t'_n)), \\
& \quad \text{where } (\Delta, t_k)_{\Phi, s}^\Lambda = (\Delta_k, t'_k), \\
& \quad \text{and } \Delta' = \bigcup_k \Delta_k \\
(\Delta, Z^n_i(a_1, \ldots, a_n))_{\Phi, s}^\Lambda & = (\Delta \cup \Delta', Z_i) \text{ if leftmost occurrence of } Z^n_i \text{ in } s \\
& \quad \text{where } \Delta' = \{a \# Z_i \mid a \in \Lambda \setminus (a_1, \ldots, a_n)\} \\
(\Delta, Z^n_i(b_1, \ldots, b_n))_{\Phi, s}^\Lambda & = (\Delta \cup \Delta', \Psi(\phi_s(Z^n_i), (b_1, \ldots, b_n)) \cdot Z_i) \text{ otherwise}, \\
& \quad \text{where } \Delta' = \{b \# Z_i \mid b \in \Lambda \setminus \phi_s(Z^n_i)\}
\end{align*}
\]

For the examples in this and next sections, \( X, Y, Z, \ldots \) range over the meta-variables instead of \( Z^n_i \), without loss of generality. This notation is closer to nominal rewriting notation and we also find it more readable. We write \( Z^n_i \) when providing definitions to follow the standard CRS notation.

**Example 6.4.** The CRS meta-term

\[
f([a]X, [b]X)
\]

is translated as the closed nominal term

\[
a \# X, b \# X \vdash f([a]X, [b]X).
\]

The use of closed nominal rewriting by the translation means that less freshness constraints are produced than by using (standard) nominal rewriting. However, some freshness must be preserved to be able to translate successfully back to CRSs when needed. For example:

\[
f([a]X(a), [b]X(b))
\]

is a CRS meta-term that produces the closed nominal translation

\[
b \# X \vdash f([a]X, [b](a \ b)) \cdot X
\]

where \( b \# X \) is necessary to preserve the property of closedness in order to translate the nominal term back to CRSs (see Definition 3.2).
This improved version of the translation is also able to deal with exotic terms which the previous translation in [14] could not handle correctly. Below, we offer an example of such a term and its translation to NRSs.

**Example 6.5.** The translation of the CRS meta-term

\[ [a][b]f([c]Z(b, c, a), [d]Z(a, b, d)) \]

to a closed NRS term is

\[ d\#Z \vdash [a][b]f([c]Z, [d](c a)(c b)) \]

A naive translation of the pair of argument lists into swappings would have modified the action of the permutation, that is, the solution \( \pi = (b \ a)(c \ b)(a \ d) \) by the translation definition in [14] is incorrect, since \( \pi(b) = c \) and \( \pi(c) = a \).

The translation function \( \mathcal{C}_r(\cdot) \) for the right-hand side of a CRS rule, when applied to a closed meta-term \( r \), produces \( (\Delta_r, \llbracket r \rrbracket_{\Phi_i}) \), where subterms of the form \( Z^n_i(t_1, \ldots , t_n) \) in \( r \) are replaced by \( \pi\cdot Z^n_i(\Phi_r(Z^n_i)_{t\mapsto [\pi]}_{\Phi_i}) \) such that, for the cases where \( t_k \in (t_1, \ldots , t_n) \) is not a variable in \( r \) (since the right rule can have any sort of CRS term within the argument list), the translation to NRS results in an explicit atom substitution \( [\Phi_l(Z^n_i)_{t\mapsto [\pi]}_{\Phi_i}] \) as defined below in Definition 6.13. On the other hand, a swapping \( (\Phi_l(Z^n_i)_{t\mapsto t_k}) \) is added to \( \pi \) where \( t_k \) is a variable occurring in \( r \) (i.e. \( t_k \) is a variable occurring bound above \( Z^n_i \) by definition of a closed CRS meta-term). Also \( \Delta_r \) contains fresh atoms for each bound variable occurring in the term.

It is important to remark that function \( \Psi \) is not applicable here. CRS systems are considered with the usual naming conventions, that is, different bound variables are used in each abstraction. Hence, there are no clashes among variable names both in \( \phi_l(Z^n_i) \) and other occurrences of \( Z^n_i \) in the right-hand side meta-term.

The original notation for explicit atom substitution given in [14] is kept for our translation model, where \( t[a \rightarrow s] \) is an abbreviation for \( \text{sub}([a]t, s) \) representing an explicit substitution. We provide the formal definition below and the rules of explicit substitution in Section 6.2 (see Definition 6.13).

**Definition 6.6 (Right Translation).** Let \( t \) be a closed CRS meta-term, \( \Phi_l \) the function defined in Definition 6.1 applied to a CRS meta-term \( l \). Then \( \mathcal{C}_r(t) = (\Delta_r, \llbracket t \rrbracket_{\Phi_i}) \) where

\[ \Delta_r = \{ a_k \#Z^n_i \mid a_k \text{ occurs bound above } Z^n_i \text{ in } t \} \]

and \( \llbracket t \rrbracket_{\Phi_i} \) is defined by:

\[
\begin{align*}
[a]_{\Phi_i} &= a \\
[a[s]_{\Phi_i} &= [a[s]_{\Phi_i} \\
[f(t_1, \ldots , t_n)]_{\Phi_i} &= f([t_1]_{\Phi_i}, \ldots , [t_n]_{\Phi_i}) \\
[Z^n_i(t_1, \ldots , t_n)]_{\Phi_i} &= \pi \cdot Z^n_i(\Phi_r(Z^n_i)_{t\mapsto [\pi]}_{\Phi_i}) \end{align*}
\]

and

\[
\pi = (\Phi_r(Z^n_i)_{t\mapsto j_1}) \cdot \ldots \cdot (\Phi_r(Z^n_i)_{t_k \mapsto j_k}) \text{, and}
\]

\[ j_1 \ldots j_k, m_1 \ldots m_k \in \{1, \ldots , n\}, \ t_{j_1}, \ldots , t_{j_k} \in A(t) \]

Similar to Definition 6.3, we take into account abstracted variables which do not appear in the meta-application.

Note that, compared with the translation given in [14], we generate less freshness constraints and we have also improved the flow when translating back and forth between CRS and NRS rules, by converting explicit substitution among atoms into swappings added to \( \pi \) (see Example 6.11 and further examples in Section 7).
Next, we prove that Definitions 6.3 & 6.6 produce closed nominal meta-terms, separately, and also as part of the translation of a CRS rule.

**Lemma 6.7.** Let \( t \) be the left-hand side of a CRS rule following Barendregt’s naming convention (i.e., variable names are pairwise distinct), \( \Phi \) the function in Definition 6.1 and \((\varnothing, t)_{Z_i}^\Delta = (\Delta, t')\) its translation as in Definition 6.3. Then \( \Delta \vdash t' \) is a closed nominal term-in-context.

**Proof.** Assume \( Z_i^n(a_1, \ldots, a_n) \), \( Z_i^0(b_1, \ldots, b_n) \) are two occurrences of the same meta-variable in \( t \) along with their respective list of distinct variables (possibly empty, thus \( Z_i^0 \)), then their translation results in \( \pi_a \cdot Z_i \), \( \pi_b \cdot Z_i \).

Hence, for \( \Delta \vdash t' \) to be a closed term-in-context (see Definition 2.9) we must prove the following:

1. no free atoms occur in \( t' \),
2. any pair \( \pi_a \cdot Z_i \), \( \pi_b \cdot Z_i \) occurring in \( t' \) satisfy conditions 2 & 3 in Definition 2.9

The first condition holds because, by definition, CRS rules are closed and the translation does not introduce new variables. For the second condition, we must consider three possible cases: the case where the list of atoms is empty, the case where one of the occurrences is the rightmost, and finally the case where none of the occurrences is the leftmost. No other cases are possible.

If the list of atoms is empty (thus \( Z_i^0 \)), both \( \text{support}(\pi_a) \), \( \text{support}(\pi_b) = \varnothing \) in \( \pi_a \cdot Z_i \), \( \pi_b \cdot Z_i \) by definition of the translation function where for each variable \( a \in \Lambda \) in the recursive call \((\Delta, Z_i^0)_{\Phi_i}^\lambda \) (i.e., any abstraction occurring above the meta-variable), \( a \# Z_i \) is added to \( \Delta \). Hence the property of closedness holds for this case.

If one of the occurrences is the leftmost in \( t \), for instance \( Z_i^n(a_1, \ldots, a_n) \), then its translation is \( \pi_a \cdot Z_i \) with \( \text{support}(\pi_a) = \varnothing \) and \( \Phi_i(Z_i^n) = [a_1, \ldots, a_n] \) where \( (a_1, \ldots, a_n) \# Z_i \not\in \Delta \). On the other hand, \( Z_i^n(b_1, \ldots, b_n) \) is translated as \( \pi_b \cdot Z \) with \( \Psi(\Phi_i(Z_i^n), (b_1, \ldots, b_n)) = \pi_b \) and \( b_k \# Z_i \in \Delta \) such that \( b_k \in \Lambda \setminus \phi_i(Z_i^n) \).

This means that \( (a_1, \ldots, a_n) \) may appear unabstracted in a substitution for \( Z_i \) but not any of \( \{b_1, \ldots, b_n\} \) distinct from \( \{a_1, \ldots, a_n\} \). If so, \( (a_1, \ldots, a_n) \) are all abstracted above \( \pi_a \cdot Z_i \) since the meta-term is closed and \( (\pi_a(a_1) = b_1, \ldots, \pi_a(a_n) = b_n) \) thus \( (a_1, \ldots, a_n) \) are also abstracted above \( \pi_b \cdot Z_i \). For any other atom that may appear in \( Z_i \), the atom is unabstracted above both occurrences. Hence the property also holds for this case.

For the case where none of the occurrences is the leftmost, there exists a leftmost one \( Z_i^n(c_1, \ldots, c_n) \) and \( \Phi_i(Z_i^n) = [c_1, \ldots, c_n] \). Therefore the translation of both occurrences is \( \pi_a \cdot Z_i \) with \( \{c_1, \ldots, c_n, a_1, \ldots, a_n\} \in \text{support}(\pi_a) \) and \( \pi_b \cdot Z_i \) with \( \{c_1, \ldots, c_n, b_1, \ldots, b_n\} \in \text{support}(\pi_b) \) where \( a_k, b_k \# Z_i \in \Delta \) for any \( a_k \in \Lambda_a \setminus \phi_i(Z_i^n) \) and \( b_k \in \Lambda_b \setminus \phi_i(Z_i^n) \) with \( \Lambda_a, \Lambda_b \) the set of variables bound over meta-applications \( Z_i^n(a_1, \ldots, a_n) \) and \( Z_i^n(b_1, \ldots, b_n) \), respectively.

This means that any of \( \{c_1, \ldots, c_n\} \) may appear unabstracted in \( Z_i \) but not any of \( \{a_1, \ldots, a_n, b_1, \ldots, b_n\} \) distinct from \( \{c_1, \ldots, c_n\} \). If so, \( (\pi_a(c_1) = a_1, \ldots, \pi_a(c_n) = a_n) \) and \( (\pi_b(c_1) = b_1, \ldots, \pi_b(c_n) = b_n) \) by Definition 6.2 hence \( (c_1, \ldots, c_n) \) are bound above both \( \pi_a \cdot Z_i \), \( \pi_b \cdot Z_i \) respectively. For any other atom that may appear in \( Z_i \), the atom is unabstracted above both occurrences.

Hence \( \Delta \vdash t' \) is a closed nominal term as expected. \( \square \)

**Lemma 6.8.** Let \( t \) be a closed CRS meta-term following the variable naming convention, \( \Phi \) the function defined in Definition 6.1 such that for each meta-variable \( Z_i^n \) in \( t \), \( Z_i^n \in
dom(Φs) for any meta-term s such that Φs(Z^n) = [a_1, ..., a_n]. Then the translation of t, Cr(t) = (Δ_t, [t]_Φ_s), is also a closed nominal term-in-context.

Proof. By induction over the structure of t and the fact that any bound variable appearing in the CRS term abstracts a fresh atom in its nominal counterpart by definition of Δ_t. For the meta-variable case, we have proved this particular case in Lemma 6.7, where none of the occurrences is the leftmost.

Remark 6.9 (CRS Term Translation). For any CRS term t, t is also a nominal ground term, trivially, since there are no meta-variables.

6.2. Transforming CRS Rules. In this section we show how CRS rules can be converted into closed NRS rules.

Definition 6.10. We define the translation of the CRS rule \( R = l \Rightarrow r \) as \( \mathcal{R} = \mathcal{C}^R(l, r) = (\Delta_l, l) \Rightarrow (\Delta_r, [r]_\Phi_l) = \Delta \vdash l' \rightarrow r' \) with \( \Delta = \Delta_l \cup \Delta_r \).

We give some examples to illustrate the definition.

Example 6.11. The translation of the \( \beta \)-rule shown in Example 2.12 according to Definition 6.10 is

\[ \vdash \text{app}(\text{lam}([a]Z), Z') \rightarrow Z[a \mapsto Z']. \]

Now, consider the CRS differentiation operator rule as taken from [14]:

\[ \text{diff}([a]\sin(Z(a))) \Rightarrow [b]\text{mult}(\text{app}(\text{diff}([c]Z(c)), b), \cos(Z(b))). \]

The translation of this rule is

\[ b\#Z, c\#Z \vdash \text{diff}([a]\sin(Z)) \rightarrow [b]\text{mult}(\text{app}(\text{diff}([c](a c) \cdot Z), b), \cos((a b) \cdot Z)) \]

where the freshness conditions are needed to preserve closedness, and mappings of atoms on the right-hand side are transformed into permutations.

Further examples can be found in Section 7.

Lemma 6.12. Let \( R = l \Rightarrow r \) be a CRS rule. If \( \Delta \vdash l' \rightarrow r' \) is its translation according to Definition 6.10 then \( \Delta \vdash l' \rightarrow r' \) is a closed nominal rewrite rule.

Proof. By definition of a CRS rule, meta-variables in r appear also in l. The rule is closed because, by Definition 6.10, \( \Phi_l \) (i.e. function \( \Phi \) applied to the left-hand side rule) is shared by the translation functions for both sides of the rule.

Let us denote by \( \mathcal{R} \) the nominal rewriting system obtained by translating all the rules of the CRS \( R \) according to Definition 6.10 and adding the rules for explicit substitution, where \( \text{sub}([a]s, t) \) is used to define the substitution of the atom \( a \) by \( t \) in \( s \), and is abbreviated \( s[a \mapsto t] \). We recall below the rules for \( \text{sub} \), taken from [14].
Definition 6.13 (Explicit Substitution Rules).

\[
\begin{align*}
(\sigma_{\text{var}}) & \quad a[a \to X] \quad \to \quad X \\
(\sigma_{t}) & \quad a \# Y \vdash Y'[a \to X] \quad \to \quad Y \\
(\sigma_{f}) & \quad (f X)[a \to Y] \quad \to \quad fX[a \to Y] \\
& \text{for each } f \text{ in } \Sigma \\
(\sigma_{\text{prod}}) & \quad (X_1, \ldots, X_n)[a \to Y] \quad \to \\
& \quad (X_1[a \to Y], \ldots, X_n[a \to Y]) \\
(\sigma_{\text{abs}}) & \quad b \# Y \vdash ([b]X)[a \to Y] \quad \to \\
& \quad [b](X[a \to Y])
\end{align*}
\]

Definition 6.14 (Normal Form of a Term-in-context). We denote by \(n_f \sigma(\Delta \vdash t)\) the normal form of a term-in-context \(\Delta \vdash t\), with respect to the rules in Definition 6.13 which is uniquely defined modulo \(\alpha\)-equivalence as proved in [14].

Lemma 6.15 (Correctness of Substitution). Let \(t, s\) be terms in a CRS \(R\) (and therefore also in nominal system \(\mathcal{R}\)). Then \(n_f \sigma(t[a := s]) \approx_{\alpha} t[a := s]\) where \(t[a := s]\) denotes the term obtained by substituting (using the higher-order substitution of the CRS) \(a\) by \(s\) in \(t\).

Proof. The proof by induction is omitted since it is standard in explicit substitutions. \qed

We are now ready to prove the correctness of the translated reduction rule.

Theorem 6.16 (Translation of CRS Rewrite Steps). Let \(R = l \Rightarrow r\) be a CRS rule. Let \(u\) be a CRS term.

If \(u \Rightarrow_R v\) then \(\vdash u \vdash^+_{\mathcal{R} \cup \sigma} v\) using \(\mathcal{R} = \mathcal{C}^\mathcal{R}(l, r)\) and the explicit substitution rules.

Proof. \(\mathcal{R} = \mathcal{C}^\mathcal{R}(l, r) = \nabla \vdash l' \to r'\) by Definition 6.10 which is a closed NRS by Lemma 6.12. Since \(u, v\) are terms in CRS \(R\), they are also ground terms in NRS \(\mathcal{R}\) by Remark 6.9. If \(u \Rightarrow_R v\) then there exists \(p, \sigma\) such that \(u|_p = l \sigma\) where \(p\) is a position in \(u\), \(\sigma\) a CRS substitute where \(\text{dom}(\sigma) \subseteq MV(l)\) and \(v = u[r \sigma]|_p\) in \(R\). Then, it is also the case that \(u|_p = l' \sigma'\) by Remark 6.9. \(\nabla \sigma' = \emptyset\) in \(\mathcal{R}\), thus we must prove that \(u[r' \sigma']|_p \to^* v\). This is done by nominal matching and induction on \(r\).

The interesting case is when \(r\) is a meta-term containing some meta-variable \(Z_i^m\), then \(\Phi_i(Z_i^m) = [a_1, \ldots, a_n]\) by Definition 6.11 and \([Z_i^m(t_1, \ldots, t_n)]_{\phi_k} = (a_{j_1} t_{j_1}) \cdots (a_{j_k} t_{j_k}) \cdot Z_i[a_{m_1} \mapsto [t_{m_1}]_{\phi_1}, \ldots, a_{m_k} \mapsto [t_{m_k}]_{\phi_1}]\) where \(j_1, \ldots, j_k, m_1, \ldots, m_k \in \{1 \ldots n\}\) and \(t_{j_1}, \ldots, t_{j_k} \in A(r)\) by definition of the translation function.

Then if \(\sigma(Z_i^m) = \lambda a_{m_1} \cdots \lambda a_{m_k}: s\) it is the case that \(\sigma')(Z_i) = s([a_{m_1} \mapsto [t_{m_1}]_{\phi_1}, \ldots, a_{m_k} \mapsto [t_{m_k}]_{\phi_1}]\).

Therefore, by Lemma 6.15 \(r \sigma \approx_{\alpha} n_f \sigma(r' \sigma')\), concluding that \(u \Rightarrow_R v\) with \(v = u[r \sigma]|_p\) implies \(\vdash u \vdash^+_{\mathcal{R} \cup \sigma} v\) where \(v = u[n_f \sigma(r' \sigma')]|_p\). \qed

We have designed an algorithm that correctly transforms CRS rules into NRS closed rules. It improves over the function defined in [14] in two ways: we have fixed a bug in the translation of zero-ary meta-variables in the scope of abstractions (see Example 6.4) the translation function given in [14] outputs the NRS term \(\vdash [a]X, [b]X\) which is not closed and permits the capture of atoms \(a, b\), and by using closed rewriting (see Definition 2.10) we are able to simulate the variable naming convention without adding extra freshness constraints, as shown in Example 6.11.

Next, we provide examples where both transformations are applied (NRSs to CRSs and back).
7. Examples

After describing the tools required to translate nominal systems to CRS (see section 4) and back to NRSs (see section 6), we dedicate this section to transform two examples between formalisms by means of the implementation found in [10].

7.1. Prenex Normal Form. We present here a translation back to NRSs by application of Definition 6.10 to the CRS rules obtained in Example 4.4. Beforehand we have applied the usual naming convention in rules (renaming bound variable $a$ to $b$ on the right-hand side):

$$b\#P, b\#Q \vdash \text{and}(P, \text{forall}([a]Q)) \rightarrow \text{forall}([b]\text{and}(P,(ab)\cdot Q))$$

$$b\#P, b\#Q \vdash \text{and}(\text{forall}([a]Q), P) \rightarrow \text{forall}([b]\text{and}(ab\cdot Q, P))$$

$$b\#P, b\#Q \vdash \text{or}(P, \text{forall}([a]Q)) \rightarrow \text{forall}([b]\text{or}(P,(ab)\cdot Q))$$

$$b\#P, b\#Q \vdash \text{or}(\text{forall}([a]Q), P) \rightarrow \text{forall}([b]\text{or}(ab\cdot Q, P))$$

Notice that a matching $\sigma = [Q \mapsto a]$ with the left rule leads to the expected result when applied to the right rule by means of the swapping $(a \cdot b)$. Also, the freshness condition $a\#P$ in the initial set of NRS rules (see Example 2.7) is converted here to $b\#P$ because of the variable convention applied beforehand. $b\#Q$ is added to the set of freshness conditions for the rules to remain closed. This does not alter the semantics when translated back to CRSs, take for instance the first NRS rule

$$b\#P, b\#Q \vdash \text{and}(P, \text{forall}([a]Q)) \rightarrow \text{forall}([b]\text{and}(P,(ab)\cdot Q))$$

By applying Definition 4.2, it translates to the CRS rule

$$\text{and}(P, \text{forall}([a]Q(a))) \rightarrow \text{forall}([b]\text{and}(P,Q(b)))$$

as expected.

7.2. Simulating $\beta$-reduction. Consider the $(\beta_{\text{lam}})$ rule given in Example 4.5. First, we apply Barendregt’s convention to the CRS rule so that each bound variable is distinct, obtaining:

$$\text{app}(\text{lam}([a]\text{lam}([b]X(a,b))), Y) \Rightarrow \text{lam}([d]\text{app}(\text{lam}([c]X(c,d)), Y))$$

Its translation to NRSs is the following:

$$d\#Y, d\#X, c\#X \vdash \text{app}(\text{lam}([a]\text{lam}[b]X,Y) \rightarrow \text{lam}[d]\text{app}(\text{lam}[c](a \cdot c)(b \cdot d)\cdot X, Y)$$

And when translated back to CRSs by means of Definition 4.2, we obtain the same CRS rule as expected.
8. Conclusions and Future Work

We have shown two extensions of first-order rewriting, CRSs and NRSs, to be closely related. The main differences are in the meta-language used, NRSs do not rely on the \( \lambda \)-calculus, employing instead permutations of atoms and a freshness relation to axiomatise \( \alpha \)-equivalence. Also NRS rules are more general than CRS rules in that unabstracted atoms may occur in rules. On the other hand, CRSs rely on the \( \lambda \)-calculus to generate its rewrite relation and CRS rules are closed by definition.

We have shown that despite these differences it is possible to translate between these formalisms. We have given a translation function which transforms the class of closed nominal rewriting systems into CRS systems. We have shown some non-trivial examples to support our work; see [10, 9] for a Haskell implementation.

Moreover, existing translation algorithms between CRSs and \( HRSs \) [34], \( \rho \)-calculus [3] and \( ERS \) [18] allow transformations from NRSs to any of these systems and vice versa.

Although previous work has been done on translating nominal syntax to higher-order syntax [26] and back to NRSs [6], our work differs from [26] by focusing on a syntax-directed mapping of NRS terms to meta-terms, extended to rules and therefore preserving the rewriting relation, which is key to the translation of properties such as confluence and termination.

We have also corrected and improved a previous algorithm translating CRSs to NRSs originally found in [14]. This means that now we have a mechanism to export results on termination of rewriting from one framework to the other.

Since nominal terms have good algorithmic properties, we could translate CRSs to NRSs in order to take advantage of existing nominal procedures (i.e., orderings, completion) then transfer back results. Another suggestion would be extending the nominal typing system to the (untyped) CRSs. This is left for future work.

Acknowledgements. We thank Elliot Fairweather and Christian Urban for many helpful discussions, and Jamie Gabbay for providing the macro for \( \mathcal{U} \). We also thank the anonymous referees for their valuable comments.

References


