Nominal Rewriting with Name Generation: Abstraction vs. Locality

Maribel Fernández and Murdoch J. Gabbay
Dept. of Computer Science
King's College London
Strand, London WC2R 2LS,UK
maribel,jamie@dcs.kcl.ac.uk

ABSTRACT

Nominal rewriting extends first-order rewriting with Gabbay-Pitts abstractors: bound entities are named, matching respects α-conversion and can be directly implemented thanks to the use of freshness constraints. In this paper we study two extensions to nominal rewriting. First we introduce a И quantifier for modelling name generation and restriction. This allows us to model higher-order functions involving local state, and has also applications in concurrency theory. The second extension introduces new constraints in freshness contexts. This allows us to express strategies of reduction and has applications in programming language design and implementation. Finally, we study confluence properties of nominal rewriting and its extensions.

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*lambda calculus and related systems*

General Terms

Theory

Keywords

Binders, α-conversion, first and higher-order rewriting, name generation, locality, confluence.

1. INTRODUCTION

Motivation.

Terminology in computer science features many words for the process of hiding variable names: *binding*, *abstraction*, *scope*, *restriction*; as well as related terms quantifying over variables such as *closed*, *open*, *linear* (i.e. in a name), and so on. Canonical examples 'in nature' of what we intend these terms to model are: λ-calculus abstraction $\lambda x.t$ [2], π-calculus restriction $\nu a.P$ [17], 'being closed' in the sense of 'having no free variables', as in closed reduction strategies [8], 'not free in' as in 'x is not free in $\lambda x.x$', and similarly 'is free in'.

This paper proposes a single unified framework encompassing the notions of '**abstraction**', '**restriction**', '**closed**', and '**is free in**' — Extended Nominal Rewriting Systems. We prove some technically nontrivial theorems which usefully extend previous work [6], but we consider the main contribution of this paper that we give these notions formal meaning side-by-side in a single well-integrated system, such that dependencies and formal structure can easily be explored.

Why choose these extensions in particular? Abstraction and 'is not free in' were our starting point from previous work [6]; we extend results of that system in this paper. 'Closed' and 'is-in' are motivated by work on reduction strategies (in particular, closed reduction in the λ-calculus [8]). Restriction expresses a π-calculus reaction $\nu[n]X \to \text{И}n.X$ where И indicates (informally, for the moment) that n must be fresh for all variables in the context in which the rewrite is introduced. Similar phenomena arise in programming languages, such as *gensym* in LISP, *unit ref* in ML, and many other forms of dynamic allocation both in the foundational and applied literatures.

Technical overview.

Our framework is a variation of Nominal Rewriting Systems (NRSs, see [6]). In particular our system is first-order (see [5]); this means terms are labelled trees and substitution is just cutting and pasting on them[1]. We write identity of terms $\equiv$.

Because our object of study here is names and binding, we have to be a little careful about our notation. We follow [10, 20, 6] and treat names of object-level syntax (which may be bound) as a special class of constants a, b, c which we call **atoms**. We reserve **variable** for the identifiers X, Y, Z that represent unknown terms. X may be substituted for by cutting and pasting, but cannot be bound.[2]

[1]For comparison, higher-order rewriting systems — e.g. Combinatory Reduction Systems [13]; Higher-order Rewrite Systems [15]; Expression Reduction Systems [11]; and the rewriting calculus [3, 4] — build in a notion of binding by quotienting terms by α-equivalence. This would make an analysis like ours more difficult, since the issues of binding and capture-avoiding substitutions are defined outside the system.

[2]Atoms and variables correspond to variables and metavari-

PPDP'05, July 11–13, 2005, Lisbon, Portugal.

Since names are constants, abstraction is a binary term-former $[a]s$. The first element is an atom (we can use types to insist on this or build the syntax that way; we choose the latter for simplicity). Dependencies of variables on atoms are implicit; for example, the β-rule of the λ-calculus is written

$$app(\lambda([a]M), N) \quad \to \quad subst([a]M, N)$$

where *subst* is an explicit substitution operator, with rules such as:

$$\begin{array}{lll} a\#Y \vdash & subst([a]Y, X) & \to \; Y \\ & subst([a]a, X) & \to \; X \\ b\#Y \vdash & subst([a]\lambda([b]X), Y) & \to \; \lambda([b]subst([a]X, Y)) \end{array}$$

Here X may be instantiated with any term unless stated otherwise by a **freshness constraint** $a\#X$ 'a is **fresh for** X' representing the promise to only instantiate with terms such that a is not free. Freshness judgements $a\#t$ formalise 'not free'; they are inductively defined, and may depend on assumptions $a\#Z$ for Z occurring in t.

α-equivalence $s \approx_\alpha t$ is a relation which must be deduced since terms are not quotients up to α-equivalence. Terms may mention variables X so α-equivalence is deduced relative to freshness assumptions $a\#X$.

Write $Symb1 \Rrightarrow Symb2$ to mean 'the definition of $Symb2$ in this paper depends on the definition of $Symb1$, and not vice versa'; then $\equiv \Rrightarrow \# \Rrightarrow \approx_\alpha$. All so-called 'nominal' techniques are motivated by the claim that this separation gives better control of names and binding than alternatives; e.g. higher-order techniques put all this (and sometimes also β-equivalence) into identity $\equiv$; and first-order techniques, which stop at $\equiv$, leave the user to handle everything else. See [6], and also [23, 22, 20, 10].

How does restriction $\mathsf{N}a.s$ differ from abstraction $[a]s$? ...much as λ from the λ-calculus relates to ν from the π-calculus: We *do not* want $[a][a]s \approx_\alpha [a]s$, following the example of $\lambda x.\lambda x.s \neq \lambda x.s$. We *do* want $\mathsf{N}a.\mathsf{N}a.s \approx_\alpha \mathsf{N}a.s$, following the π-calculus example of $\nu a.\nu a.P = \nu a.P$. We *do* want $[a]a \approx_\alpha [b]b$, following $\lambda x.x = \lambda y.y$, but we *do not* want $\mathsf{N}a.a \approx_\alpha \mathsf{N}b.b$, since N generates fresh names each time. We *do not* want $[a][b]s \approx_\alpha [b][a]s$ but we *do* want $\mathsf{N}a.\mathsf{N}b.s \approx_\alpha \mathsf{N}b.\mathsf{N}a.s$ (it does not matter what order we generate names).

We can think of abstraction $[a]s$ as an abstractor in the familiar λ sense, and of restriction $\mathsf{N}a.s$ as a tag marking a name as local (and restricted) to a local scope. We may accordingly call a **local** if bound by a N, and **global** otherwise. N does not abstract at the level of syntax, i.e. $\approx_\alpha$. Yet the π-calculus has scope extrusion rules which do rename to avoid capture. We resolve this with a *rewrite rule* (not a syntactic identity or equivalence), called **rule** (F), which we informally write here as $\mathsf{N}a.s_a \to \mathsf{N}b.s_b$. Thus $\mathsf{N}a.s$ generates a name which *is* a, but the term can rewrite itself to generate a name which *is* b, if necessary. Compare this with abstraction $[a]s$ which involves a name which we *call* a but which we could have called something different.

On the level of contexts, this paper extends Nominal Rewriting with extra judgements $\bullet t$ meaning intuitively $a\#t$ always, $a\in t$ meaning intuitively $a\#x$ does *not* hold, and $a@s$ meaning intuitively that if a is unabstracted in s, it is always restricted under some N (we want $a\#s$ to imply $a@s$ so we use $\#$ in the definition of $@$: in notation introduced above there is a clear dependency $\# \Rrightarrow @$). Rewrite rules may insist some conditions are satisfied for the rewrite to take place — we say the rules are **triggered**, so these extensions allow us to express some nice reduction strategies.

ables respectively in Combinatory Reduction Systems, though the notion of substitution is different; see [6] for more details.

We give inductive definitions of fresh, restricted, and α-equivalence, and derive algorithms to check them. We also generalise the nominal unification algorithm [23] to solve equations between terms containing restricted names. We study confluence of rewriting and give a critical pair lemma; this extends results from a previous paper [6]. We also prove that orthogonal systems are confluent, which specialises to a result for unextended nominal rewriting which we had not proved in the previous paper.

The authors are not aware of any comparative analysis of abstraction and restriction in the literature. Momigliano and Pfenning [18] add a finer-grained function type to the simply-typed λ-calculus to say 'f does use its argument', which resembles our 'is-in' predicate and provides independent motivation for considering it. Miller and Tiu [16] consider a quantifier ∇ which is similar to N and partly motivated it, in that it is recognisably a restriction operator, though our N acts on the level of terms whereas ∇ is a quantifier in a logic and so acts on the level of predicates. The same comments apply to the N quantifier of Nominal Logic [20, 9].

N is related to and directly modelled on the ν operator of Pitts and Stark's ν-calculus [21] and ν the π-calculus restriction operator [17]. Apart from obvious differences between a simply typed λ-calculus, a process calculus, and a rewriting formalism, ν and N differ in their treatment of generated names: ν really is a binder and N is a restrictor, such that $\mathsf{N}a.a \not\equiv \mathsf{N}b.b$. However, with our notion of rewriting and with the right rewrite rules, N models the behaviour of both.

Overview of the paper.
Section 2 presents nominal signatures and extended nominal terms. Section 3 defines α-equivalence as a logical notion first, then gives an algorithm to test α-equality. Nominal matching and unification algorithms are given in Section 4. Section 5 defines rewriting on extended nominal terms, and Section 6 studies confluence. Section 7 introduces closedness constraints and reduction strategies. We conclude the paper in Section 8.

2. EXTENDED NOMINAL TERMS

2.1 Signatures and terms

A **nominal signature** Σ is a set of **function symbols** typically written f. We do not assume that functions have sorts or arities in this paper (the original presentation [6] does but we shall not need them here).

So for example, a nominal signature for a fragment of ML has function symbols:

`var   app   lam   let   letrec`

Fix some signature Σ. Fix a countably infinite set $\mathcal{X}$ of **term variables** X, Y, Z; these represent meta-level unknowns. Fix a distinct countably infinite set $\mathcal{A}$ of **atoms** a, b, c, n, x; these represent object-level variable symbols. Consistent with later notation for terms, we write $a \equiv a$ to denote identity of atoms. We assume that Σ, $\mathcal{X}$ and $\mathcal{A}$ are pairwise disjoint. We use A, B to denote sets of atoms.

A **swapping** is a pair of atoms, which we write $(a\ b)$. **Permutations** π are lists of swappings, generated by the grammar

$$\pi ::= \mathbf{Id} \mid (a\ b)\cdot\pi$$

We call **Id** the **identity permutation**. We call a pair of a permutation π and a variable X a **moderated variable** and write it $\pi\cdot X$.

Extended nominal terms, also called **tagged terms** or just **terms** for short, are trees labelled by function symbols, atoms, etc. (i.e. syntax in the standard sense, later we call this the term's **skeleton**) but also annotated at every node with a finite set of atoms, which we shall call **tags**. Terms are generated by the grammar

$$\begin{array}{lll} s,t & ::= & \mathsf{N}A.u \\ u,v & ::= & a \mid \pi\cdot X \mid (t_1,\ldots,t_n) \mid [a]t \mid (ft) \end{array}$$

where tags A may be empty. A term such that every tag is empty is in essence ordinary syntax and is isomorphic to a(n unextended) nominal term [6]. We may write $\mathsf{N}\emptyset.u$ more succinctly as u, and similarly we may write $\mathbf{Id}\cdot X$ as X.

Terms are called respectively **atoms**, **moderated variables** (or just variables for short), **tuples**, **abstractions** and **function applications**.

In the clause for tuples, n is called the **length** of the tuple and may equal 0 in which case we have the empty tuple $()$. We omit the brackets when n is 1 if there is no ambiguity. Note that f may be applied to the empty tuple in which case we may write $f()$ as just f.

We denote $V(t)$ the set of variables that occur in the term t. **Ground terms** are terms without variables, that is $V(t) = \emptyset$. A ground term may still contain atoms, for example a or $\mathsf{N}\{a\}.()$ are ground terms and X is not.

An abstraction $[a]t$ is intended to represent t with a bound, and accordingly we call occurrences of a **abstracted** (or bound) and unabstracted occurrences **unabstracted** (or free). We do *not* work modulo α-conversion of abstracted atoms, so syntactic identity $\equiv$ is *not* modulo α-equivalence; for example, $[a]a \not\equiv [b]b$. α-equivalence $\approx_\alpha$ is a logical notion constructed on top of $\equiv$ using a notion of context which we shall define soon.

A tagged term $\mathsf{N}\{a\}.u$ is intended to represent u with a local name a. We call occurrences of a **restricted** or **local**, and unrestricted occurrences **unrestricted** or **global**. As with abstraction, we do not work up to α-conversion of tags.

Abstraction is inherited from previous work [23, 6], tags are new to this paper. We give some examples:

- $\mathsf{N}\{a\}.a$ is a restricted copy of a. In future, we drop the set curly brackets and write $\mathsf{N}a.a$. Note that $\mathsf{N}a.a \not\equiv \mathsf{N}b.b$.
- $\mathsf{N}a.b$ is b with a local name a (which is not used, the reader might like to consider it '**garbage**'). $\mathsf{N}a.b$ is not syntactically identical to $\mathsf{N}c.b$, $\mathsf{N}b.b$, or b.
- $\mathsf{N}a,b.a$ generates a and b and discards b. Because the tags are sets, $\mathsf{N}a,b.a \equiv \mathsf{N}b,a.a$. However, $\mathsf{N}a,b.a \not\equiv \mathsf{N}a,b.b$.
- $\mathsf{N}a.(\mathsf{N}\emptyset.a, \mathsf{N}\emptyset.a)$ generates a and makes the pair (a,a).
- $(\mathsf{N}a.a, \mathsf{N}a.a)$ generates two local names which are both (by coincidence) 'a'.
- $\mathsf{N}a.(a, \mathsf{N}a.a)$ is a tuple which generates locally a and places it in the first component. The second component is also a, but this is a different copy in the scope of its own local restrictor. $\mathsf{N}a.(a,\mathsf{N}a.a) \not\equiv (\mathsf{N}a.a, \mathsf{N}a.a)$.
- Using the ML signature mentioned above, we can write $(\mathtt{lam}[a]\mathsf{N}b.(\mathtt{app}(\mathtt{var}(a), \mathtt{app}(\mathtt{var}(b), X))))$ which we will abbreviate $\lambda[a]\mathsf{N}b.a(bX)$. This term can be used to represent a function which has an argument a and generates a local name b (i.e. `ref` in ML).

2.2 Substitution and swapping

Substitution of an unknown X for a term s in a term t is a function that satisfies:

$$(\mathsf{N}A.a)[X\mapsto s] \equiv \mathsf{N}A.a \quad (\mathsf{N}A.(ft))[X\mapsto s] \equiv \mathsf{N}A.(f(t[X\mapsto s]))$$
$$(\mathsf{N}A.(t_1,\ldots,t_n))[X\mapsto s] \equiv \mathsf{N}A.(t_1[X\mapsto s],\ldots,t_n[X\mapsto s])$$
$$(\mathsf{N}A.[a]t)[X\mapsto s] \equiv \mathsf{N}A.[a](t[X\mapsto s])$$
$$(\mathsf{N}A.\pi\cdot X)[X\mapsto \mathsf{N}B.s] \equiv \mathsf{N}(A\cup\pi\cdot B).\pi\cdot s.$$

A **substitution** is generated by the grammar

$$\sigma ::= \mathbf{Id} \mid \sigma[X\mapsto s].$$

σ has an action on terms given elementwise from the definition above and satisfying $s\mathbf{Id} \equiv s$. We write substitutions postfix as just shown, and write $\circ$ for composition of substitutions: $t(\sigma\circ\sigma') \equiv (t\sigma)\sigma'$.

The intuition of a moderated variable $\pi\cdot X$ is that we would like to rename the atoms in X, but we do not yet know what X is. If a substitution $[X\mapsto s]$ instantiates X, as in the last clause of the definition above, we apply π to s. A permutation has an action on terms, denoted $\pi\cdot t$, defined in the natural way from the component swappings. The action of swappings on terms is defined inductively by:

$$(a\ b)\cdot\mathsf{N}A.u \equiv \mathsf{N}(a\ b)\cdot A.(a\ b)\cdot u$$
$$(a\ b)\cdot a \equiv b \qquad (a\ b)\cdot b \equiv a \quad \text{and} \quad (a\ b)\cdot c \equiv c\ (c \not\equiv a,b)$$
$$(a\ b)\cdot(ft) \equiv (f(a\ b)\cdot t)$$
$$(a\ b)\cdot(t_1,\ldots,t_n) \equiv ((a\ b)\cdot t_1,\ldots,(a\ b)\cdot t_n)$$
$$(a\ b)\cdot[n]t \equiv [(a\ b)\cdot n](a\ b)\cdot t$$
$$(a\ b)\cdot\pi\cdot X \equiv ((a\ b)\cdot\pi)\cdot X.$$

where the action of swappings on sets of atoms is pointwise.

For example, $(a\ b)\cdot\lambda[a]\mathsf{N}b,c.abX \equiv \lambda[b]\mathsf{N}a,c.ba(a\ b)\cdot X$.

We write π^{-1} for the permutation obtained by reversing the swappings in π; e.g. if π is $(a\ b)\cdot(b\ c)$ then $\pi^{-1}\cdot a \equiv c$.

2.3 Positions

A **skeletal position** p is generated by the grammar

$$p ::= \epsilon \mid p.n$$

where n is a number. A **position** is a pair of a skeletal position and a finite (possibly empty) set of atoms A. We write $s|_{p,A}$ for the subterm of s at position (p,A).

Skeletal positions are usually called positions, we assume the reader is familiar with [5, 12]; see also [1]. 'Skeletal' refers to the tag-less 'skeleton' of a term: so ϵ is the root position; 1 indicates going under a function symbol or an abstraction $[a]s$; i refers to ith position in $(s_1,\ldots,s_n)$; permutations are ignored. Having arrived at a certain place p in the skeleton, 'A' means remove the atoms A from the tag at that (skeletal) position. We omit a formal definition but give examples:

If $s \equiv \text{И}a,b.(\text{И}a.X,Y)$ then $s|_{\epsilon,\emptyset} \equiv s$, $s|_{\epsilon,\{a\}} \equiv \text{И}b.(\text{И}a.X,Y)$, $s|_{\epsilon,\{a,b\}} \equiv \text{И}\emptyset.(\text{И}a.X,Y)$, and $s|_{\epsilon,\{a,b,c\}}$ is undefined. $s|_{\epsilon.1,\emptyset} \equiv \text{И}a.X$, $s|_{\epsilon.1,\{a\}} \equiv X$, $s|_{\epsilon.2,\emptyset} \equiv Y$, and $s|_{\epsilon.3,\emptyset}$ is not defined. If $s' \equiv [a]t$ then $s'|_{\epsilon,\emptyset} \equiv s'$ and $s'|_{\epsilon.1,\emptyset} \equiv t$.

We write $s[t]_{p,A}$ for the result of replacing the subterm of s at (p,A) by t, merging the tags. We may omit A when it is empty, writing $s[t]_p$ instead of $s[t]_{p,\emptyset}$. For example if s is as above, then $s[Z]_\epsilon \equiv Z$, $s[Z]_{\epsilon,\{a\}} \equiv \text{И}a.Z$, $s[\text{И}b.Z]_{\epsilon,\{a\}} \equiv \text{И}a,b.Z$, $s[\text{И}c.Z]_{\epsilon,\{b\}} \equiv \text{И}b,c.Z$.

3. ALPHA-EQUIVALENCE

Syntactic equality $s \equiv t$ is a structural (rather than logical) fact. α-equivalence is a logical notion, which we define below.

3.1 A logical presentation

Constraints are generated by the grammar

$$P, Q, C ::= a\#t \mid a@t \mid s \approx_\alpha t.$$

We call $\#$ a **freshness** predicate, $@$ a **locality** predicate, and $\approx_\alpha$ an **equality** predicate. They will be defined formally below. The intended interpretation of

- $a\#t$ is "if a occurs in t then each occurrence is either in a tag, or abstracted". For example, $a\#b$, $a\#\text{И}a.b$, not $a\#\text{И}a.a$, however $a\#\text{И}a.[a]a$. We sometimes write $a,b\#s$ instead of $a\#s, b\#s$.

- $a@t$ is "if an occurrence of a is unabstracted in t then it is in a tag". For example, $a@\text{И}a.b$ and $a@\text{И}a.a$, $a@[a]a$, but not $a@a$. We abbreviate $a@s, b@s$ as $a,b@s$.

- $s \approx_\alpha t$ is "s and t are α-equivalent" (see below).

$a\#s$ is specified by a system of natural-deduction rules as follows, where a,b denote different atoms:

$$\frac{a\#u}{a\#\text{И}A.u} \qquad \frac{}{a\#b} \qquad \frac{a\#s}{a\#fs} \qquad \frac{a\#s_1 \ \cdots \ a\#s_n}{a\#(s_1,\ldots,s_n)}$$

$$\frac{}{a\#[a]s} \qquad \frac{a\#s}{a\#[b]s} \qquad \frac{\pi^{\text{-}1}{\cdot}a\#X}{a\#\pi{\cdot}X} \qquad \frac{a\#a}{P}\,(\#\bot)$$

This definition generalises the one in [6]: to calculate $\#$ we simply ignore tags.

$a@s$ is specified by deduction rules as follows (again a,b denote different atoms):

$$\frac{}{a@\text{И}A.u}\,a \in A \qquad \frac{a@u}{a@\text{И}A.u}\,a \notin A$$

$$\frac{}{a@b} \qquad \frac{a@s}{a@fs} \qquad \frac{a@s}{a@[b]s} \qquad \frac{}{a@[a]s}$$

$$\frac{a@s_1 \ \cdots \ a@s_n}{a@(s_1,\ldots,s_n)} \qquad \frac{\pi^{\text{-}1}{\cdot}a@X}{a@\pi{\cdot}X} \qquad \frac{a\#X}{a@X}\,(\#@) \qquad \frac{a@a}{P}\,(@\bot)$$

All rules except for $(\#@)$, $(\#\bot)$ and $(@\bot)$ are echoed in algorithmic form in the next subsection.

Finally we define $\approx_\alpha$ inductively as follows, where again a,b denote different atoms, and $ds(\pi,\pi')$ denotes the **difference set** of the two permutations:

$$ds(\pi,\pi') \ \stackrel{\text{def}}{=} \ \{n \mid \pi{\cdot}n \neq \pi'{\cdot}n\}$$

$$\frac{u \approx_\alpha v \qquad B\backslash A@u \quad A\backslash B@v}{\text{И}A.u \approx_\alpha \text{И}B.v}$$

$$\frac{}{a \approx_\alpha a} \qquad \frac{s_1 \approx_\alpha t_1 \ \cdots \ s_n \approx_\alpha t_n}{(s_1,\ldots,s_n) \approx_\alpha (t_1,\ldots,t_n)} \qquad \frac{s \approx_\alpha t}{fs \approx_\alpha ft}$$

$$\frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \qquad \frac{s \approx_\alpha (a\ b){\cdot}t \quad a\#t}{[a]s \approx_\alpha [b]t} \qquad \frac{ds(\pi,\pi')\#X}{\pi{\cdot}X \approx_\alpha \pi'{\cdot}X}$$

For example, $ds((a\ b),\mathbf{Id}) = \{a,b\}$, so (using the rules above) we can deduce $(a\ b){\cdot}X \approx_\alpha X$ from assumptions $a\#X$ and $b\#X$ and we also have as expected $[a]a \approx_\alpha [b]b$. Note that, consistent with intuitions discussed in the Introduction, $\text{И}a.b \approx_\alpha b$, and $\text{И}a,b.a \approx_\alpha \text{И}a.a$, but note that $\text{И}a.a \not\approx_\alpha \text{И}b.b$.

Say constraints of the form $a\#a$, $a\#X$, $a@X$, and $a@a$ are **reduced**. We write Δ, ∇, Γ for sets of reduced constraints, we may call them **contexts**. If there are no constraints of the form $a\#a$ or $a@a$ in Δ we say it is **consistent**.

Call a set Pr of constraints a **problem**. We write $\Delta \vdash Pr$ when proofs of P exist for all $P \in Pr$, using elements of Δ as assumptions, and we say that Δ **entails** Pr. Since the rules above decompose syntax, an algorithm to check entailment can be easily built using this logical presentation. We give an operational definition of the predicates $\#$, $@$ and $\approx_\alpha$ in the next subsection.

We now state some properties of $\#$, $@$ and $\approx_\alpha$ which are needed later. In particular, we show that $\approx_\alpha$ is a congruence.

Lemma 3.1. *If $\nabla \vdash a\#t$ then $\nabla \vdash a@t$.*

Proof. We examine the derivation rules for $@$ and $\#$ and see that a proof of $a\#t$ can be transformed rule-for-rule to a proof of $a@t$. We may use $(\#@)$ to deduce $a@X$ from some $a\#X \in \nabla$. $\square$

Lemma 3.2. *Suppose ∇ is a reduced context and t, t' are terms with or without root tags.*

1. *$\nabla \vdash a\#\pi{\cdot}t$ if and only if $\nabla \vdash \pi^{\text{-}1}{\cdot}a\#t$.*
2. *$\nabla \vdash a@\pi{\cdot}t$ if and only if $\nabla \vdash \pi^{\text{-}1}{\cdot}a@t$.*
3. *$\nabla \vdash a\#\text{И}A.u$ if and only if $\nabla \vdash a\#\text{И}B.u$.*
4. *If $\nabla \vdash a\#t$ and $\nabla \vdash t \approx_\alpha t'$ then $\nabla \vdash a\#t'$.*
5. *If $\nabla \vdash a@t$ and $\nabla \vdash t \approx_\alpha t'$ then $\nabla \vdash a@t'$.*
6. *If $ds(\pi,\pi') = \emptyset$ then $\nabla \vdash \pi{\cdot}t \approx_\alpha \pi'{\cdot}t$ for any π, π'.*

Proof. The first, second, and third parts are by routine inductions on derivations.

The fourth part is by mostly routine induction on derivations. We use the first part for the case of $[a]s \approx_\alpha [b]t$, and the third part for the case of $\text{И}A.u \approx_\alpha \text{И}B.v$.

The fifth part follows from the second part in a similar way. We consider two example cases:

Suppose $\nabla \vdash x@\text{И}A.u$ and $\nabla \vdash \text{И}A.u \approx_\alpha \text{И}A'.u'$ are derivable.Then $\nabla \vdash c@u$ for every $c \in A' \setminus A$, and $\nabla \vdash c@u'$ for every $c \in A \setminus A'$, and $\nabla \vdash u \approx_\alpha u'$. There are now three cases:

- If $x \in A'$ then $\nabla \vdash x@\text{И}A'.u'$ and we are done.

- If $x \in A$ and $x \notin A'$ then $\nabla \vdash x@u'$ and so $\nabla \vdash x@\text{И}A'.u'$ and we are done.

- If $x \notin A$ and $x \notin A'$ then $\nabla \vdash x@u$ and $\nabla \vdash u \approx_\alpha u'$ and we use the inductive hypothesis.

Suppose $\nabla \vdash x@[a]t$ and $\nabla \vdash [a]t \approx_\alpha [a']t'$. Then $\nabla \vdash t \approx_\alpha (a\ a')\cdot t'$ and $\nabla \vdash a\#t'$.

- If $x \equiv a'$ we are done.
- If $x \equiv a$ we use the previous lemma to derive $\nabla \vdash a@t'$ using the derivation of $\nabla \vdash a\#t'$, then extend to a derivation of $\nabla \vdash a@[a']t'$.
- If $x \notin \{a, a'\}$ we use the inductive hypothesis to deduce $\nabla \vdash x@(a\ a')\cdot t'$, the second part of this lemma to deduce $\nabla \vdash x@t'$, and finally extend to a derivation of $\nabla \vdash x@[a']t'$ as required.

The last part is by induction on t. $\square$

LEMMA 3.3. *For fixed ∇, $\approx_\alpha$ is an equivalence relation and $\nabla \vdash s \approx_\alpha t$ implies $\nabla \vdash \pi\cdot s \approx_\alpha \pi\cdot t$.*

PROOF. Reflexivity is easy. Transitivity and preservation of $\approx_\alpha$ by permutations are proved simultaneously, by induction on derivations, using Lemma 3.2. Symmetry then follows by induction on derivations and Lemma 3.2. $\square$

Note an immediate corollary of Lemma 3.2 (parts 1 and 2) and Lemma 3.3:

COROLLARY 3.4. *$\Delta \vdash Pr$ if and only if $\Delta \vdash \pi\cdot Pr$.*

The following properties will be used later on.

COROLLARY 3.5.
1. *$\Delta \vdash s \approx_\alpha \pi^{-1}\cdot\pi\cdot s'$ if and only if $\Delta \vdash s \approx_\alpha s'$.*
2. *$\Delta \vdash s \approx_\alpha \pi\cdot s'$ if and only if $\Delta \vdash \pi^{-1}\cdot s \approx_\alpha s'$.*
3. *$\Delta \vdash ds(\pi,\pi')\#s$ if and only if $\Delta \vdash \pi\cdot s \approx_\alpha \pi'\cdot s$.*
4. *$\Delta \vdash A \setminus A'@s$ and $\Delta \vdash A' \setminus A@s$ then $\Delta \vdash \reflectbox{N}A.s \approx_\alpha \reflectbox{N}A'.s$.*
5. *For fixed Δ, $\approx_\alpha$ is a congruence. Thus, if $\Delta \vdash t \approx_\alpha t'$ then $\Delta \vdash s[t]_p \approx_\alpha s[t']_p$.*
6. *$\#$, $@$ and $\approx_\alpha$ are preserved by substitution:*
   - *$\Delta \vdash a\#t$ and $\Delta' \vdash \Delta\sigma$ implies $\Delta' \vdash a\#t\sigma$. Similarly for $@$.*
   - *$\Delta \vdash t \approx_\alpha t'$ and $\Delta' \vdash \Delta\sigma$ implies $\Delta' \vdash t\sigma \approx_\alpha t'\sigma$.*

PROOF. The first four parts are by routine inductions using the previous results. Part 5 is by induction on the syntax of s and uses the previous cases. Part 6 is by induction on t and by induction on $\approx_\alpha$. $\square$

3.2 An algorithmic presentation

We now specify **simplification rules** on sets of constraints, which will be used to check them for derivability. We will later extend these rules to solve matching problems between left-hand sides of rewrite rules and terms (in anticipation we use l for terms in the simplification rules for $\approx_\alpha$). In the rules we assume a, b denote different atoms.

$$
\begin{array}{rcll}
a\#\reflectbox{N}A.u, Pr & \Longrightarrow & a\#u, Pr & \\
a\#b, Pr & \Longrightarrow & Pr & \\
a\#fs, Pr & \Longrightarrow & a\#s, Pr & \\
a\#(s_1,\dots,s_n), Pr & \Longrightarrow & a\#s_1,\dots,a\#s_n, Pr & \\
a\#[b]s, Pr & \Longrightarrow & a\#s, Pr & \\
a\#[a]s, Pr & \Longrightarrow & Pr & \\
a\#\pi\cdot X, Pr & \Longrightarrow & \pi^{-1}\cdot a\#X, Pr & \pi \not\equiv \mathbf{Id}
\end{array}
$$

$$
\begin{array}{rcll}
a@\reflectbox{N}A.u, Pr & \Longrightarrow & Pr & a \in A \\
a@\reflectbox{N}A.u, Pr & \Longrightarrow & a@u, Pr & a \notin A \\
a@b, Pr & \Longrightarrow & Pr & \\
a@fs, Pr & \Longrightarrow & a@s, Pr & \\
a@(s_1,\dots,s_n), Pr & \Longrightarrow & a@s_1,\dots,a@s_n, Pr & \\
a@[b]s, Pr & \Longrightarrow & a@s, Pr & \\
a@[a]s, Pr & \Longrightarrow & Pr & \\
a@\pi\cdot X, Pr & \Longrightarrow & \pi^{-1}\cdot a@X, Pr & \pi \not\equiv \mathbf{Id}
\end{array}
$$

$$
\begin{array}{rcl}
\reflectbox{N}A.u \approx_\alpha \reflectbox{N}B.v, Pr & \Longrightarrow & B\backslash A@u,\ A\backslash B@v, u \approx_\alpha v,\ Pr \\
a \approx_\alpha a, Pr & \Longrightarrow & Pr \\
(l_1,\dots,l_n) & \Longrightarrow & l_1 \approx_\alpha s_1, \dots, \\
\approx_\alpha (s_1,\dots,s_n), Pr & & \quad l_n \approx_\alpha s_n, Pr \\
fl \approx_\alpha fs, Pr & \Longrightarrow & l \approx_\alpha s, Pr \\
[a]l \approx_\alpha [a]s, Pr & \Longrightarrow & l \approx_\alpha s, Pr \\
[b]l \approx_\alpha [a]s, Pr & \Longrightarrow & (a\ b)\cdot l \approx_\alpha s, a\#l, Pr \\
\pi\cdot X \approx_\alpha \pi'\cdot X, Pr & \Longrightarrow & ds(\pi,\pi')\#X, Pr
\end{array}
$$

These rules define a reduction relation on problems: We write $Pr \Longrightarrow Pr'$ if Pr' is obtained from Pr by applying a simplification rule. We denote by $\stackrel{*}{\Longrightarrow}$ its transitive and reflexive closure.

For example, the problem $\{\reflectbox{N}a.a \approx_\alpha \reflectbox{N}\emptyset.a,\ \reflectbox{N}\emptyset.a \approx_\alpha \reflectbox{N}\emptyset.a\}$ reduces to $\{a@a,\ a \approx_\alpha a\}$ using the first rule for $\approx_\alpha$ twice. Other examples are:

$$a\#(X,[a]Y) \stackrel{*}{\Longrightarrow} a\#X \qquad a\#fa \stackrel{*}{\Longrightarrow} a\#a$$
$$a\#((a\ b)\cdot X, (b\ c)\cdot Y) \stackrel{*}{\Longrightarrow} b\#X, a\#Y$$

These reduction rules are derived from the deduction rules in the last subsection, but there are differences; we omit a rule $a@X, Pr \Longrightarrow a\#X, Pr$, which would correspond to $(\#@)$, and also rules corresponding to $(\#\bot)$ and $(@\bot)$. If we didn't, the important lemmas below would not hold.

LEMMA 3.6. *The simplification rules above are confluent and strongly normalising (reduction order does not matter).*

PROOF. By Newman's Lemma [19] we need only show termination, because there are no critical pairs. The rules form a hierarchical system in the sense of [7], from which it follows that if the first two groups of rules are terminating and non-duplicating (they are) and do not use in the right-hand side any symbol defined in the third group (i.e. equality $\approx_\alpha$; they do not), then if the rules defining the equality symbol satisfy the general recursive scheme, then the whole system is terminating. The general recursive scheme requires that recursive calls in right-hand sides use strict subterms of the left-hand side arguments, and this is the case (permutations are ignored). $\square$

Write $\langle Pr\rangle_{nf}$ for the unique normal form of Pr, and $\langle P\rangle_{nf}$ for $\langle\{P\}\rangle_{nf}$, i.e. the result of simplifying it as much as possible. We will say that an equality $u \approx_\alpha v$ is **clashing** when u and v have different term constructors at the root (e.g. $[a]s \approx_\alpha (t,t')$, $X \approx_\alpha ft$, $a \approx_\alpha X$, etc.) or they are two different atoms (e.g. $a \approx_\alpha b$) or two different variables (e.g. $\pi\cdot X \approx_\alpha \pi'\cdot Y$), or applications with different function symbols (e.g. $ft \approx_\alpha gs$).

LEMMA 3.7 (NORMAL FORMS). *$\langle a\#s\rangle_{nf}$ consists of reduced freshness constraints and $\langle a@s\rangle_{nf}$ consists of reduced locality constraints. $\langle s \approx_\alpha t\rangle_{nf}$ may contain any mixture of reduced freshness constraints, reduced locality constraints, and clashing equalities.*

PROOF. Any non-reduced freshness or locality constraint can be simplified using the rules in the first two groups. Similarly, a non-clashing equation can be simplified with a rule in the third group. $\square$

Therefore $\langle Pr\rangle_{nf}$ can always be partitioned into $\Delta \cup Eq$ where Δ is a (possibly empty) context and Eq a (possibly empty) set of clashing equalities Eq.

LEMMA 3.8. *Assume $Pr \Longrightarrow Pr'$. Then*

$$\Gamma \vdash Pr \text{ if and only if } \Gamma \vdash Pr'.$$

PROOF. By inspection of the rules. $\square$

THEOREM 3.9. *Let $\langle Pr\rangle_{nf} = \Delta \cup Eq$ where Δ is a context and Eq is a set of clashing equalities.*

1. *If Δ is consistent then:*
   $\Delta \vdash Pr$ if and only if $Eq = \emptyset$.

2. *$\Delta \vdash P$ for all P if and only if Δ is inconsistent.*

PROOF. The first part is proved by induction on the length of the derivation $Pr \stackrel{*}{\Longrightarrow} \langle Pr\rangle_{nf}$, using Lemma 3.8: $\Delta \vdash Pr$ if and only if $\Delta \vdash Eq$; the result follows from the fact that if $u \approx_\alpha v$ is clashing then there is no consistent Δ such that $\Delta \vdash u \approx_\alpha v$.

For the second part, we use the rules $(\#\bot)$ and $(@\bot)$. $\square$

COROLLARY 3.10. *Let Γ be a consistent context.*

$$\Gamma \vdash Pr \text{ if and only if } \langle Pr\rangle_{nf} = \Delta \text{ and } \Gamma \vdash \Delta.$$

PROOF. Consequence of Theorem 3.9 and Lemma 3.8. $\square$

4. UNIFICATION

A **unification problem** Pr is a problem as previously defined but replacing the equality constraint $s \approx_\alpha t$ by the unification constraint $s\ {}_?{\approx}_?\ t$. A **solution** to a unification problem Pr is a pair (Γ, σ) of a **consistent context** and a **substitution** such that $\Gamma \vdash Pr'\sigma$ where Pr' is obtained from Pr by changing unification predicates into equality predicates, and $Pr'\sigma$ is the problem obtained by applying the substitution σ to the terms in Pr'. If there is no consistent context satisfying this property, we say that Pr is **unsolvable**. We write $\mathcal{U}(Pr)$ for the set of unification solutions to Pr.

We want some algorithm returning *most general* solutions. Thus we must decide when (Γ, σ) is **more general** than (Γ', σ'). This notion is complicated by **(a)** the freshness context Γ and **(b)** the identity $\text{И}a.\text{И}a.s \equiv \text{И}a.s$.

Thus **(a)** $(\{a\#Y, b\#Y\}, [X\mapsto(a\ b)\cdot Y])$ should be equivalent to $(\{a\#X, b\#X\}, [X\mapsto Y])$ (since $a, b\#Y \vdash (a\ b)\cdot Y \approx_\alpha Y$), and $(\{a@Y\}, [X\mapsto \text{И}a.Y])$ should similarly be equivalent to $(\{a@Y\}, [X\mapsto Y])$ (because $a@Y \vdash Y \approx_\alpha \text{И}a.Y$). Also **(b)** we expect $(\{\}, [X\mapsto a])$ to be a more general solution than $(\{\}, [X\mapsto \text{И}a.a])$, *if* the problem in question is $\text{И}a.X\ {}_?{\approx}_?\ \text{И}a.a$ (because the tag 'swallows' any $\text{И}a$ on X).

We need some notation. Write $\Delta \vdash Pr\sigma \approx_\alpha Pr\sigma'$ when for each $P \in Pr$, $\Delta \vdash P\sigma \approx_\alpha P\sigma'$, and in addition for all X, $X\sigma \equiv X$ if and only if $X\sigma' \equiv X$ ('σ and σ' have the same domain').

Then we define: $(\Gamma, \sigma) \leq_{Pr} (\Gamma', \sigma')$ when there exists a substitution σ'' such that $\Gamma' \vdash \Gamma\sigma''$ and $\Gamma' \vdash Pr(\sigma \circ \sigma'') \approx_\alpha Pr\sigma'$. This defines a partial order on solutions, call it the **instantiation ordering** (reflexivity of $\leq$ is direct, transitivity follows from last part of Corollary 3.5). Since Pr is fixed we usually drop the subscript on the $\leq$. Say a solution is **principal** when it is smaller than or equal to all other solutions to that problem. A solution (Γ, σ) is **idempotent** when for all X, $\Gamma \vdash X\sigma \approx_\alpha X(\sigma \circ \sigma)$.

We will now show that every solvable unification problem has a principal idempotent solution. The algorithm is derived from the simplification rules from the previous section, enriched with **instantiating** rules, labelled with substitutions, with priority over the other rules. (The conditions in the instantiating rules are usually called **occurs check**.)

$$\text{И}A.\pi\cdot X\ {}_?{\approx}_?\ \text{И}B.u, Pr \stackrel{X\mapsto\pi^{-1}\cdot\text{И}B\backslash A.u}{\Longrightarrow} Pr[X\mapsto\pi^{-1}\cdot\text{И}B\backslash A.u] \quad (X \notin V(u))$$

$$\text{И}B.u\ {}_?{\approx}_?\ \text{И}A.\pi\cdot X, Pr \stackrel{X\mapsto\pi^{-1}\cdot\text{И}B\backslash A.u}{\Longrightarrow} Pr[X\mapsto\pi^{-1}\cdot\text{И}B\backslash A.u] \quad (X \notin V(u))$$

We do not *solve* freshness or restriction constraints — for this we would need instantiating rules for them. For rewriting it suffices to solve equality constraints, and require that the solution satisfy the freshness and restriction constraints in the context of the rule.

LEMMA 4.1. *The simplification rules given in the previous section (where we replace $\approx_\alpha$ by ${}_?{\approx}_?$), together with the instantiation rules, are confluent and strongly normalising.*

PROOF. Standard: at each step either the number of unsolved variables X in the problem decreases, or a non-reduced constraint is simplified towards a reduced constraint. $\square$

These rules specify a unification algorithm: to solve a problem Pr, we will apply the rules until we obtain an irreducible problem. We write $\langle Pr\rangle_{sol} = (Pr', \sigma)$ if Pr' is the (unique) normal form of Pr and σ is obtained by concatenating the labels in some reduction sequence from Pr to Pr'. We can characterise normal forms:

LEMMA 4.2 (UNIFICATION NORMAL FORMS). *$\langle a\#s\rangle_{sol}$ consists of reduced freshness constraints and $\langle a@s\rangle_{sol}$ consists of reduced locality constraints. $\langle s\ {}_?{\approx}_?\ t\rangle_{sol}$ may contain any mixture of reduced freshness constraints, reduced locality constraints, and clashing equalities such that if one side is a variable the other side contains this variable.*

PROOF. Similar to Lemma 3.7: Any non-reduced freshness or locality constraint or non-clashing equality can be simplified as previously. Equalities of the form $\text{И}A.\pi\cdot X\ {}_?{\approx}_?\ \text{И}B.u$ or $\text{И}B.u\ {}_?{\approx}_?\ \text{И}A.\pi\cdot X$ where $X \notin V(u)$ can be reduced using an instantiation rule. $\square$

Therefore we can decompose $\langle Pr\rangle_{sol}$ as a triple (Δ, Eq, σ) where Δ is a context, Eq a set of equations, and σ a substitution.

We will show that the unification algorithm correctly checks whether a problem is solvable or not, and moreover it computes a principal, idempotent solution, if one exists. In the proof we will use the following lemmas:

LEMMA 4.3. 1. *If Eq is a non-empty set of unification constraints in normal form then it has no solution.*

2. *If Δ is an inconsistent context then it has no solution.*

PROOF. By Lemma 4.2, the equalities in Eq are clashing or fail the occur check. Since no consistent context can entail an instance of such an equality, Eq is not solvable.

The second part is directly by definition of solution, and the fact that from a consistent context we cannot derive an inconsistent one. $\square$

LEMMA 4.4. *$(\pi{\cdot}t)\sigma \equiv \pi{\cdot}(t\sigma)$.*

PROOF. π acts top-down and accumulates on moderated variables. σ acts bottom up on the variable symbols in the moderated variables. The two operations commute. A formal proof is easy by induction on syntax. $\square$

LEMMA 4.5. *If $\Delta \vdash (\text{И}A.\pi{\cdot}X)\sigma \approx_\alpha (\text{И}B.u)\sigma$ (for $X\notin u$) then writing $\sigma' \equiv [X{\mapsto}\pi^{-1}\text{И}B\backslash A.u]\sigma$ we have also $\Delta \vdash (\text{И}A.\pi{\cdot}X)\sigma' \approx_\alpha (\text{И}A.\pi{\cdot}X)\sigma$.*
Similarly for $\Delta \vdash (\text{И}B.u)\sigma \approx_\alpha (\text{И}A.\pi{\cdot}X)\sigma$.

PROOF. By routine manipulations up to Δ, exploiting the previous lemma to rearrange brackets. $\square$

LEMMA 4.6 (PRESERVATION OF SOLUTIONS). *Assume $Pr \stackrel{\theta}{\Longrightarrow} Pr'$.*

1. *If $(\Gamma,\sigma) \in \mathcal{U}(Pr)$ then $(\Gamma,\sigma) \in \mathcal{U}(Pr')$ and $\Gamma \vdash Pr(\theta \circ \sigma) \approx_\alpha Pr\sigma$.*

2. *If $(\Gamma,\sigma) \in \mathcal{U}(Pr')$ then $(\Gamma,\theta\circ\sigma) \in \mathcal{U}(Pr)$.*

PROOF. The first part follows using the previous lemma.
For the second part, suppose $\Gamma \vdash Pr'\sigma$ and suppose $\theta = [X{\mapsto}\pi^{-1}{\cdot}\text{И}A.u]$ so that

$$(\{\text{И}A.u \approx_\alpha \text{И}A.u\} \cup Pr') = Pr[X{\mapsto}\pi^{-1}\text{И}A.u].$$

It is easy to verify that $\Gamma \vdash \text{И}A.u \approx_\alpha \text{И}A.u$, so $\Gamma \vdash Pr(\theta \circ \sigma)$. $\square$

COROLLARY 4.7. *Let Pr be a unification problem, and $\langle Pr\rangle_{sol} = (\Delta, Eq, \sigma)$.*
$\mathcal{U}(Pr) \neq \emptyset$ if and only if Δ is consistent and $Eq = \emptyset$.

PROOF. By induction on length for the derivation $Pr \stackrel{*}{\Longrightarrow} \langle Pr\rangle_{sol}$. We use Lemma 4.3 and Lemma 4.6 part 1 for the "only if", and Lemma 4.6 part 2 for the "if". $\square$

LEMMA 4.8. *If $\langle Pr\sigma\rangle_{nf} = \Delta$ then $\langle Pr\rangle_{nf} = Pr'$ and $\langle Pr'\sigma\rangle_{nf} = \Delta$ for some Pr'.*
If $\langle Pr\rangle_{sol} = (\Delta,\emptyset,\sigma)$ then $\langle Pr\sigma\rangle_{nf} = \Delta$.

PROOF. The first part is a consequence of the confluence of $\Longrightarrow$.
The second part is proved by induction on the derivation $Pr \stackrel{*}{\Longrightarrow} \langle Pr\rangle_{sol}$. A step $\text{И}A.\pi{\cdot}X \;{}_?{\approx}_?\; \text{И}B.u, Pr \stackrel{X\mapsto\pi^{-1}\cdot\text{И}B\backslash A.u}{\Longrightarrow} Pr[X{\mapsto}\pi^{-1}{\cdot}\text{И}B \setminus A.u]$ is replaced by
$\text{И}B.u \approx_\alpha \text{И}B.u, Pr[X{\mapsto}\pi^{-1}{\cdot}\text{И}B \setminus A.u] \stackrel{*}{\Longrightarrow} Pr[X{\mapsto}\pi^{-1}{\cdot}\text{И}B \setminus A.u]$. $\square$

THEOREM 4.9 (PRINCIPALITY). *Let Pr be a unification problem, and $\langle Pr\rangle_{sol} = (\Delta,\emptyset,\sigma)$ such that Δ is a consistent context.*

1. *$(\Delta,\sigma) \in \mathcal{U}(Pr)$, and*

2. *if $(\Gamma,\sigma') \in \mathcal{U}(Pr)$ then $(\Delta,\sigma) \leq (\Gamma,\sigma')$.*

PROOF. For the first part, note that $(\Delta,\mathbf{Id}) \in \mathcal{U}(\Delta)$ if Δ is a consistent context. Hence by Lemma 4.6 (and induction) $(\Delta,\sigma) \in \mathcal{U}(Pr)$.
For the second part, if $(\Gamma,\sigma') \in \mathcal{U}(Pr)$ then:

- $\Gamma \vdash Pr(\sigma \circ \sigma') \approx_\alpha Pr\sigma'$, by Lemma 4.6 part 1 (and induction). Note in particular that because of the occurs check, $X\sigma \equiv X$ or $X \notin X\sigma$, so $X\sigma \equiv X(\sigma\circ\sigma)$ always and σ is idempotent.

- $\Gamma \vdash Pr\sigma'$ by definition of solution, and $\Gamma \vdash \langle Pr\sigma'\rangle_{nf}$ by Corollary 3.10. Also, $\Gamma \vdash \langle Pr\sigma'\rangle_{nf} \approx_\alpha \langle Pr(\sigma\circ\sigma')\rangle_{nf} = \langle (Pr\sigma)\sigma'\rangle_{nf}$, and by Lemma 4.8 part 2, $\langle Pr\sigma\rangle_{nf} = \Delta$. Therefore, by Lemma 4.8 part 1, $\langle \Delta\sigma'\rangle_{nf} = \langle (Pr\sigma)\sigma'\rangle_{nf}$, and we get $\Gamma \vdash \langle\Delta\sigma'\rangle_{nf}$. Hence $\Gamma \vdash \Delta\sigma'$ by Corollary 3.10.

So (Δ,σ) is a principal and idempotent solution. $\square$

5. REWRITING

5.1 Matching problems

A **matching problem (in context)** is a pair $(\nabla \vdash l) \;{}_?{\approx}\; (\Delta \vdash s)$ where ∇, Δ are consistent contexts and l, s are extended nominal terms. The **solution** to this matching problem, if it exists, is a pair (Δ',θ) which is a principal solution in the sense of the last section to the unification problem $\nabla, l \;{}_?{\approx}_?\; s$, such that

- $X\theta \equiv X$ for $X \in V(\Delta, s)$ and

- $\Delta \vdash \Delta'$.

This definition ensures that our solution is a matching modulo α-conversion in the usual sense: since (Δ',θ) is a solution for the unification problem $\nabla, l \;{}_?{\approx}_?\; s$, the conditions above ensure that we can derive $\Delta \vdash l\theta \approx_\alpha s$. Moreover, we can also derive $\Delta \vdash \nabla\theta$. The latter shows that we have more than matching modulo α-conversion: we can use ∇ to specify constraints which must be satisfied by the matching solution. When the conditions in ∇ are satisfied we say the matching **is triggered**. (Soon, $\nabla \vdash l$ will be the left-hand side of a rewrite rule $\nabla \vdash l \to r$, and then we say the rule is **triggered**.)

For example: $(\vdash \text{И}a.a) \;{}_?{\approx}\; (\vdash \text{И}b.b)$ has no solution; $(\vdash \text{И}b.b) \;{}_?{\approx}\; (\vdash \text{И}b.b)$ has a solution $\theta = \mathbf{Id}$; and $(\vdash\text{И}a,b.X') \;{}_?{\approx}\; (\vdash\text{И}b,a.X)$ has solution $\theta = [X'{\mapsto}X]$. All of the following problems have solution $(\{a@X\},[X'{\mapsto}X])$ and show that tags are not used as term-constructors in pattern-matching, unlike abstractions (tags simply provide restriction information):

$$(\vdash\text{И}a.X') \;{}_?{\approx}\; (a\#X\vdash X)$$

$$(\vdash X') \;{}_?{\approx}\; (a\#X\vdash\text{И}a.X) \quad (\vdash X') \;{}_?{\approx}\; (a@X\vdash\text{И}a.X)$$

See [6] for more examples.

5.2 Rewrite rules

A(n extended) rewrite rule $R \equiv \nabla \vdash l \to r$ is a tuple of a consistent context ∇, and extended terms l and r such that $V(r,\nabla) \subseteq V(l)$.

Suppose $R = \nabla \vdash l \to r$ is a rewrite rule, s and t are terms, and Δ is a context. We say s **rewrites with** R **to** t **in the context** Δ, and we write $\Delta \vdash s \xrightarrow{R} t$ when:

1. $V(R) \cap V(\Delta, s) = \emptyset$ (we can assume this with no loss of generality).

2. There exists a position (p, A) in s and a matching solution (Δ',θ) to $(\nabla \vdash l) \;{}_?{\approx}\; (\Delta \vdash s|_{p,A})$.

3. $\Delta \vdash s[r\theta]_{p,A} \approx_\alpha t$.

This is essentially the same definition as for (unextended) nominal rewriting [6], modulo slight changes to account for the tags.

Write $R^{(a\,b)}$ for that rule obtained by swapping a and b in R throughout. For example, if $R \equiv b@X \vdash [a]X \to \text{И}b.(a\ b)\cdot X$ then $R^{(a\,b)} \equiv a@X \vdash [b]X \to \text{И}a.(b\ a)\cdot X$. Say a set of rewrite rules is **equivariant** when it is closed under $(-)^{(a\,b)}$ for all atoms a and b.

A rewrite system $\mathcal{R}$ is an equivariant set of rewrite rules. A set of rewrite rules gives rise naturally to a rewrite system by closing under equivariance. We shall elide this step, saying 'R has the rewrite *blah*' to mean 'the rewrite system obtained by closing R under equivariance has the rewrite *blah*'. This usage is established [6].

We now consider some simple examples, and give applications of extended nominal rewriting in the following subsection.

- The rewrite system $\{a \to a\}$ has the rewrites $n \to n$ for any n (because if $a \to a \in \mathcal{R}$ then so is $n \to n$ for any n by equivariance). It also has the rewrite $\text{И}a.a \to \text{И}a.a$, using position $(\epsilon, \{a\})$.

- The rewrite system $\{a \to \text{И}a.a\}$ has the rewrites $a \to \text{И}a.a$ and also $\text{И}a.a \to \text{И}a.a$.

- The rewrite system $\{[a]X \to \text{И}a.X\}$ has the rewrite $[a]X \to \text{И}a.X$. It also has the rewrite $b\#X \vdash [a]X \to \text{И}b.(b\ a)\cdot X$. This is because $b\#X \vdash [a]X \approx_\alpha [b](b\ a)\cdot X$.

The following is an important correctness result:

THEOREM 5.1. *If $\Delta \vdash s \xrightarrow{R} t$ and $\Delta \vdash C[t] \approx_\alpha D$, then $\Delta \vdash C[s] \xrightarrow{R} D$.*

PROOF. By unpacking the definition of rewriting above. □

5.3 Applications

Modelling name generation in the π-calculus.
A reaction system for an asynchronous π-calculus is given by the signature

$$\mathbf{in}, \mathbf{out}, \mathbf{par}, \mathbf{rep}, \nu$$

and the following rewrite rules (where we abbreviate $\mathbf{par}(s,t)$ as $(s \mid t)$, $\mathbf{in}(a,[c]t)$ as $a[c].t$, $\mathbf{out}(a,b)$ as $\overline{a}b$ and $\mathbf{rep}(t)$ as $!t$ to get the standard π-calculus notation) together with a structural congruence which includes commutativity and associativity of **par**, etc. (we omit the congruence since it is not relevant to this paper).

$$\overline{a}b \mid a[c].Y \to Y\{c{\mapsto}b\} \quad \nu[c]X \to \text{И}c.X \quad !X \to !X \mid X$$
$$a\#P \vdash P \mid (\text{И}a.Q) \to \text{И}a.(P \mid Q).$$

Here $\{c{\mapsto}b\}$ is an explicit substitution with reactions including $(X \mid Y)\{c{\mapsto}b\} \to X\{c{\mapsto}b\} \mid Y\{c{\mapsto}b\}$ and $(\nu[a]X)\{c{\mapsto}b\} \to \nu[a](X\{c{\mapsto}b\})$.

Call the last rule the **scope extrusion** rule, the first rule the **reaction rule**, and the second rule the **name-generation** rule. Let us consider the name-generation and scope extrusion rules and demonstrate in the context of extended nominal rewriting why they have these names.

Note that c is abstracted in $\nu[c]X$. Nominal rewriting, including the extension in this paper, works on terms up to provable α-equivalence; thus if $\Delta \vdash \nu[c]R \to \text{И}c.R$ is a valid rewrite and $\Delta \vdash d\#R$ is deducible, then $\Delta \vdash \nu[d](d\ c)\cdot R \to \text{И}d.(d\ c)\cdot R$ is a valid rewrite. Thus, the name-generation rule 'generates a fresh d'. И defines the scope of d and the scope extrusion rule allows us to extend it.

The fly in our ointment is that the scope extrusion rules do not allow renaming a scoped name (d, for example) to a 'fresher' d' such that, say, $d'\#P$ is provable where $d\#P$ was not. This is known as **name-clash** and it is 'unfair' since we could always have chosen d' originally.

We can add a **freshening** rewrite rule

$$F \equiv b\#P \vdash \text{И}a.P \to \text{И}b.(b\ a)\cdot P \tag{1}$$

to allow post-factum renaming. A weaker alternative is to strengthen the scope extrusion rules:

$$b\#P, Q \vdash P \mid (\text{И}a.Q) \to \text{И}b.(P \mid (b\ a)\cdot Q)$$

('Weaker', because we get rewrites of the transitive closure of F with the original scope extrusion rules, but not quite all of them.)

Name-generation in Pitts and Starks's ν-calculus.
A reaction system for the ν-calculus [21] is

$$(\lambda[a]X)Y \to X\{a{\mapsto}Y\} \quad \nu[a]X \to \text{И}a.X.$$

Again, $\{a{\mapsto}Y\}$ is an explicit substitution. Reaction rules include $a\#Y \vdash (\lambda[a]X)\{b{\mapsto}Y\} \to \lambda[a](X\{b{\mapsto}Y\})$ and $a\#Y \vdash (\nu[a]X)\{b{\mapsto}Y\} \to \nu[a](X\{b{\mapsto}Y\})$. И scope-extrudes upwards, e.g. $\vdash \lambda[a]\text{И}b.X \to \text{И}b.\lambda[a]X$.

As in the implementation of the π-calculus above we have faithfully modelled abstraction in the syntax by abstraction in the rewrite system; for example a is abstracted in $\lambda[a]s$ and $\nu[a]s$. We consider alternatives in the Conclusions.

Rewriting is a little too weak in its treatment of restricted atoms, because they cannot be renamed to avoid name-clash. Accordingly, we shall base our theory of confluence on rewriting in the presence of **rule** F from (1). We shall see this rule has a non-trivial and delicate interaction with the definitions we have set up so far, giving us precisely what we want.

6. CONFLUENCE

6.1 Uniform rewrite rules

We shall give a well-behavedness condition on rewrite rules which we call **uniformity**, but first we prove properties of $\vdash$ which give uniformity its power.

LEMMA 6.1.
- *If $\Delta \vdash Pr$ and $\Delta, \langle Pr\rangle_{nf} \vdash Pr'$ then $\Delta \vdash Pr'$.*

- *If $\Delta, \langle a\#s\rangle_{nf} \vdash a\#t$ then $\Delta, \langle a\#C[s]\rangle_{nf} \vdash a\#C[t]$. Similarly for @.*

- *If $\nabla \vdash a\#l$ then $\langle \nabla\sigma\rangle_{nf} \vdash a\#l\sigma$. Similarly for @.*

PROOF.
- By Corollary 3.10, if $\Delta \vdash Pr$ then $\Delta \vdash \langle Pr\rangle_{nf}$. We modify the deduction of $\Delta, \langle Pr\rangle_{nf} \vdash Pr'$ to replace any use of assumptions in $\langle Pr\rangle_{nf}$ with a (part of) the deduction of $\Delta \vdash \langle Pr\rangle_{nf}$.

- We work by induction on C, write X for the 'hole' in C. We consider three interesting representative cases. If $C \equiv a$ then $\langle a\#C[s]\rangle_{nf}$ is $a\#a$ and $\Delta, \langle a\#a\rangle_{nf} \vdash a\#a$

trivially. If $C \equiv (a,X)$ then $a\#a \in \langle a\#C[s]\rangle_{nf}$ so $\Delta, \langle a\#C[s]\rangle_{nf} \vdash \langle a\#C[t]\rangle_{nf}$ by $(\#\bot)$. If $C \equiv [a]X$ then $\vdash a\#C[t]$ is easily derivable.

- If $\langle\nabla\sigma\rangle_{nf}$ is inconsistent the result follows directly by rules $(@\bot)$ and $(\#\bot)$. Otherwise we use Corollaries 3.5 and 3.10.

$\square$

Part 1 of the lemma above is a very thinly disguised Cut. Part 2 is, we think, rather striking. Part 3 is perhaps less surprising.

Say a rule $R \equiv \nabla \vdash l \to r$ is **uniform** when $\nabla, \langle a@l\rangle_{nf} \vdash a@r$ for all a. Examples of uniform rules are:

$$[a]X \to \text{И}a.X \quad a\#X \vdash \text{И}a.X \to X \quad a@X \vdash \text{И}a.X \to X$$
$$a \to a \qquad a \to \text{И}b.b$$

For example consider $a \to a$. $c@a \vdash c@a$ for all $c \not\equiv a$, and $a@a \vdash a@a$ by $(@\bot)$. Consider $a \to \text{И}b.b$. $x@a \vdash x@\text{И}b.b$ is provable for $x \equiv a$ by $(@\bot)$, for $x \equiv b$, and for $x \notin \{a,b\}$. Examples of rules that are not uniform are:

$$[a]X \to X \qquad \text{И}a.X \to X \qquad a \to b$$

If *(1)* $\mathcal{R}$ is an equivariant set of uniform rewrite rules and *(2)* $F \in \mathcal{R}$ then call $\mathcal{R}$ **uniform**. Henceforth, fix some uniform rewrite system.

LEMMA 6.2. *If $\Delta \vdash s \to t$ then $\Delta, \langle a@s\rangle_{nf} \vdash a@t$ and $\Delta, \langle a\#s\rangle_{nf} \vdash a@t$ for all a.*

PROOF. If $\Delta \vdash s \to t$ then for some $R \equiv \nabla \vdash l \to r \in \mathcal{R}$ there is some position (p,A) in s and substitution σ such that $\Delta \vdash \nabla\sigma$ and $\Delta \vdash s|_{p,A} \approx_\alpha l\sigma$.

By the definition of uniformity, $\nabla, \langle a@l\rangle_{nf} \vdash a@r$. By the first and third parts of Lemma 6.1, $\Delta, \langle a@l\sigma\rangle_{nf} \vdash a@r\sigma$. By the second part of that same result and Lemma 3.2, $\Delta, \langle a@s\rangle_{nf} \vdash a@t$. We use similar reasoning for $\Delta, \langle a\#s\rangle_{nf} \vdash a@t$, only we use the first part of that lemma again, and Lemma 3.1. $\square$

In fact, we can prove something rather stronger, if we assume the context can allow us to deduce enough freshness information. Say Δ **has enough fresh atoms** with respect to some finite set of atoms A and finite set of unknowns V when there exists a set of atoms A' with at least as many elements as A such that $A \cap A' = \emptyset$ and $\Delta \vdash a'\#X$ for every $a' \in A'$ and $X \in V$.

LEMMA 6.3. *Let A be a finite set of atoms including $A(s)$, and assume Δ has enough fresh atoms with respect to the atoms and variables in A and s. If $\Delta \vdash s \to t$ then there is t' such that $\Delta \vdash s \to t \xrightarrow{F}{}^* t'$, $\Delta, \langle a@s\rangle_{nf} \vdash a\#t'$ and $\Delta, \langle a\#s\rangle_{nf} \vdash a\#t'$ for all $a \in A$.*

PROOF. As for the last lemma but using rule F to freshen atoms as illustrated in the following example. $\square$

Example: Consider the rewrite system with just the trivial rewrite rule $\emptyset \vdash X \to X$ (which is uniform). If $s \equiv \text{И}a.X$ and $\Delta \equiv b\#X$, then Δ has enough fresh atoms for $A = \{a\}$ and X, namely b, and in the notation of the lemma above, $t \equiv \text{И}a.X$ and $t' \equiv \text{И}b.(b\ a)\cdot X$.

Henceforth we suppose Δ provides us with all the fresh atoms we need. For finite terms and finite rewrite sequences, if we run out we can always try again with a larger Δ.

Uniform rewriting behaves well with respect to α-equality:

LEMMA 6.4. *If $\Delta \vdash s \to t$ and $\Delta \vdash s \approx_\alpha s'$ then $\Delta \vdash s' \to t$.*

PROOF. If the reduction takes place at position ϵ the result is trivial (because matching takes into account $\approx_\alpha$). Otherwise we proceed by induction on the definition of $\approx_\alpha$. The only interesting case is $\Delta \vdash [a]s \approx_\alpha [b]s'$. Then $\Delta \vdash s \approx_\alpha (a\ b)\cdot s'$ and $\Delta \vdash a\#s'$. Assume $\Delta \vdash [a]s \to [a]t$, then by induction $\Delta \vdash (a\ b)\cdot s' \to t$ and therefore $\Delta \vdash s' \to (a\ b)\cdot t$ because rewriting is preserved under swapping (Lemma 6.5). Hence $\Delta \vdash [b]s' \to [b](a\ b)\cdot t$ (rewriting is closed under context).

Since uniform rewriting preserves freshness constraints by Lemma 6.3, we also know that $\Delta \vdash a\#(a\ b)\cdot t$, therefore $\Delta \vdash b\#t$. Then $\Delta \vdash [b](a\ b)\cdot t \approx_\alpha [a]t$, and $\Delta \vdash [b]s' \to [a]t$ as required. $\square$

Uniformity is crucial for the result above. Consider the non-uniform rewrite system $\{(c,a) \to b\}$. Then $[a](c,a) \to [a]b$, and $[a](c,a) \approx_\alpha [b](c,b)$ but $[b](c,b) \not\to [a]b$.

LEMMA 6.5. *Rewriting is preserved by permutation: If $\Delta \vdash \pi\cdot s \to t$ then $\Delta \vdash s \to \pi^{-1}\cdot t$.*

PROOF. Using equivariance of the rewrite system and Corollary 3.4. $\square$

6.2 Critical Pairs

Fix a rewrite system $\mathcal{R}$. Write $\Delta \vdash s \to t_1, t_2$ for the appropriate pair of rewrite judgements, call it a **peak**. Suppose

1. $R_i = \nabla_i \vdash l_i \to r_i$ for $i = 1,2$ are copies of two rules in $\mathcal{R}$ such that $V(R_1) \cap V(R_2) = \emptyset$ (R_1 and R_2 could be copies of the same rule).
2. (p,A) is a position in l_1.
3. $\nabla_1, \nabla_2, l_1|_{p,A}\ {}_?{\approx}_?\ l_2$ has solution (Γ,θ) so $\Gamma \vdash l_1|_{p,A}\theta \approx_\alpha l_2\theta$ and $\Gamma \vdash \nabla_i\theta$ for $i = 1,2$.

Then call the pair of terms-in-context

$$\Gamma \vdash (r_1\theta, l_1[r_2\theta]_{p,A})$$

a **critical pair**. If $(p,A) = (\epsilon,\emptyset)$ and R_1, R_2 are copies of the same rule, or if (p,A) is the position of a variable $\pi\cdot X$ in l_1 then we say the critical pair is **trivial**.

The condition $\Gamma \vdash \nabla_i\theta$ for $i = 1,2$ is important for critical pairs to represent peaks. Rules $b\#X \vdash f(X) \to X$ and $\vdash f(b) \to c$ do not generate a peak; $f(X)\ {}_?{\approx}_?\ f(b)$ does have a solution $(\emptyset, [X \mapsto b])$, but it does not satisfy $b\#X$.

Say a uniform nominal rewrite system $\mathcal{R}$ is **locally confluent** when if $\Delta \vdash s \to t$ and $\Delta \vdash s \to t'$, then t'' exists such that $\Delta \vdash t \to^* t''$ and $\Delta \vdash t' \to^* t''$. We say such a peak is **joinable**.

Say a uniform nominal rewrite system is **confluent** when if $\Delta \vdash s \to^* t$ and $\Delta \vdash s \to^* t'$, then t'' exists such that $\Delta \vdash t \to^* t''$ and $\Delta \vdash t' \to^* t''$.

Trivial critical pairs are not necessarily joinable in NRSs. For instance with (non-uniform) rules

$$b\#X \vdash (X,b) \to c \quad \text{and} \quad f(a) \to b$$

the trivial critical pair $(c,(b,b))$ obtained by unifying $f(a)$ with $b\#X \vdash X$ is not joinable. We obtain exactly the same phenomenon with $b@X$ replacing $b\#X$.

The following result is the reason we consider uniform rules and use rule F from (1):

LEMMA 6.6. *In a uniform rewrite system trivial critical pairs are joinable.*

PROOF. Suppose there is a trivial critical pair

$$\Gamma \vdash (r_1\theta, l_1[r_2\theta]_{p,A})$$

between rules $R_i = \nabla_i \vdash l_i \to r_i$ for $i = 1, 2$, where (p, A) is the position of a variable in l_1.

The only reason we might not be able to apply R_1 in $l_1[r_2\theta]_{p,A}$ is if some freshness or restriction condition in ∇_1 is unsatisfiable after R_2, which was satisfiable before R_2. For uniform rules, Lemma 6.3 guarantees that this cannot happen (up to rule F): we can always rewrite $r_2\theta$ using F and then apply R_1. Therefore the critical pair is joinable. $\square$

Note also that thanks to Lemma 6.4, also pairs that are α-equivalent to trivial critical pairs are joinable.

This is the reason we are interested in critical pairs:

THEOREM 6.7 (CRITICAL PAIR LEMMA). *If all nontrivial critical pairs of a uniform nominal rewrite system are joinable, then it is locally confluent.*

PROOF. Suppose $\Delta \vdash s \to t_1$ and $\Delta \vdash s \to t_2$ is a peak (and as usual assume the context has enough fresh atoms). Then there exist $R_i = \nabla_i \vdash l_i \to r_i$, positions (p_i, A_i) in s, and solutions (Δ'_i, σ_i) to $(\nabla_i \vdash (s[l_i]_{p_i,A_i}, s[r_i]_{p_i,A_i}))\ {}_?{\approx}\ (\Delta \vdash (s, t_i))$ for $i = 1, 2$.

Now there are two possibilities:

1. p_1 and p_2 are in separate subtrees. Local confluence holds by a standard diagrammatic argument taken from the first-order case [1]. We need Lemma 6.4 to account for the use of α-equality.

2. p_1 is a prefix of p_2 or vice versa, we consider only the first possibility. Suppose that $p_1 = \epsilon$, the general case follows using Corollary 3.5.

   There are two cases: Either this peak is α-equivalent to an instance of a non-trivial critical pair, joinable by assumption, or it is α-equivalent to an instance of a trivial critical pair, we use Lemma 6.6 (in both cases we also use Lemma 6.4).

$\square$

We now study rule F more closely, to deal with two *problems*, before tying up the results with the Corollary below.

We call a critical pair **F-trivial** if it is either trivial in the sense defined above, or if at least one of the rewrites is F.

Problem 1. Rule F has (F-trivial) critical pairs with *any* rule $\nabla \vdash l \to r$ in which l mentions tags, simply because F can freshen them. *Solution.* The following lemma implies that F-trivial critical pairs are always joinable:

LEMMA 6.8. *If $\Delta \vdash s \xrightarrow{F} s'$ then $\Delta \vdash s' \xrightarrow{F} s$ (rule F is **reversible**).*

PROOF. By calculations involving the following easily-verified *technical fact* $b\#s \vdash (a\ b){\cdot}(b\ c){\cdot}s \approx_\alpha (a\ c){\cdot}s$. $\square$

So we can strengthen the critical pair lemma to only check joinability of critical pairs that are not F-trivial. *We may be lax* and just say '**trivial**' and '**nontrivial**' for 'F-trivial' and 'not F-trivial', e.g. in the corollary below.

Problem 2. $\mathcal{R}$ is necessarily nonterminating because F can be applied to any term with a tag, forever (given enough fresh atoms). *Solution.* $\mathcal{R}$ gives rise to an **F^*-abstract** rewrite system obtained from $\mathcal{R}$ by counting one or more instances of F in sequence as one F^*-abstract rewrite (we count $\xrightarrow{F}{}^*$ as a single 'abstract' rewrite step). Say $\mathcal{R}$ is **terminating up to** F^* when its F^*-abstract rewrite relation is terminating. By Newman's Lemma [19] we can deduce confluence of a locally confluent and F^*-terminating uniform rewrite system. *We may be lax* and write '**terminating**' for F^*-terminating.

COROLLARY 6.9. *Fix a uniform nominal rewrite system.*

1. *If nontrivial critical pairs are joinable then it is locally confluent.*

2. *If it is in addition terminating then it is confluent, and every term has a unique normal form modulo $\approx_\alpha$ and rewrites with F.*

This result holds of nominal rewrite systems in the previous sense [6], which are the special case of the system here without И.

6.3 Orthogonal Systems

We now treat a standard confluence criterion in rewriting theory [5, 12] (see also [13, 15]).

Say a rule $R \equiv \Delta \vdash l \to r$ is **left-linear** when each variable occurring in l occurs only once. For example, $a\#X, b\#X \vdash fX \to (X, X)$ is left-linear but $\vdash (X, X) \to X$ is not.

We say a uniform, nominal rewrite system with only left-linear rules and no non-trivial critical pairs is **orthogonal**.

THEOREM 6.10. *An orthogonal uniform nominal rewrite system is confluent.*

The proof occupies the rest of this section. Henceforth, we only consider uniform rewriting.

We define a **parallel reduction** relation as follows:

$$\frac{\Delta \vdash u \xrightarrow{F}{}^* v}{\Delta \vdash u \Rightarrow v}\,(\text{refl}) \qquad \frac{\Delta \vdash (s_i \Rightarrow t_i)_{1 \le i \le n}}{\Delta \vdash (s_1, \ldots, s_n) \Rightarrow (t_1, \ldots, t_n)}\,(\text{tup})$$

$$\frac{\Delta \vdash (s_i \Rightarrow t_i)_{1 \le i \le n} \quad \Delta \vdash (t_1, \ldots, t_n) \xrightarrow{R_\epsilon}\xrightarrow{F}{}^* t'}{\Delta \vdash (s_1, \ldots, s_n) \Rightarrow t'}\,(\text{tup'})$$

$$\frac{\Delta \vdash s \Rightarrow t}{\Delta \vdash [a]s \Rightarrow [a]t}\,(\text{abs}) \qquad \frac{\Delta \vdash s \Rightarrow t \quad \Delta \vdash [a]t \xrightarrow{R_\epsilon}\xrightarrow{F}{}^* t'}{\Delta \vdash [a]s \Rightarrow t'}\,(\text{abs'})$$

$$\frac{\Delta \vdash \text{И}A.u \xrightarrow{F_\epsilon}{}^* \text{И}A'.u' \quad \Delta \vdash u' \Rightarrow v' \quad \Delta \vdash \text{И}A'.v' \xrightarrow{F_\epsilon}{}^* \text{И}B.v}{\Delta \vdash \text{И}A.u \Rightarrow \text{И}B.v}\,(\text{new})$$

$$\frac{\Delta \vdash \text{И}A.u \xrightarrow{F_\epsilon}{}^* \text{И}A'.u' \quad \Delta \vdash u' \Rightarrow v' \quad \Delta \vdash \text{И}A'.v' \xrightarrow{F_\epsilon}{}^* \xrightarrow{R_\epsilon}\xrightarrow{F}{}^* t}{\Delta \vdash \text{И}A.u \Rightarrow t}\,(\text{new'})$$

$$\frac{\Delta \vdash s \Rightarrow t}{\Delta \vdash fs \Rightarrow ft}\,(\text{fun}) \qquad \frac{\Delta \vdash s \Rightarrow t \quad \Delta \vdash ft \xrightarrow{R_\epsilon}\xrightarrow{F}{}^* t'}{\Delta \vdash fs \Rightarrow t'}\,(\text{fun'})$$

$$\frac{\Delta \vdash a \xrightarrow{R_\epsilon}\xrightarrow{F}{}^* t'}{\Delta \vdash a \Rightarrow t'}\,(\text{atom'}) \qquad \frac{\Delta \vdash \pi{\cdot}X \xrightarrow{R_\epsilon}\xrightarrow{F}{}^* t'}{\Delta \vdash \pi{\cdot}X \Rightarrow t'}\,(\text{var'})$$

We have used some new notation: $\Delta \vdash s \xrightarrow{R_\epsilon} t$ means 's rewrites to t using R at position $\epsilon, \emptyset$'. In the case of rule F, $\Delta \vdash s \xrightarrow{F_\epsilon} t$ means s rewrites to t using F at position ϵ, A, for some A.

LEMMA 6.11. $\xrightarrow{F}{}^* \Rightarrow \xrightarrow{F}{}^* \subseteq \Rightarrow$.

PROOF. By induction on the derivation of $\Rightarrow$. The interesting cases are (new) and (new'); we use the technical fact from the proof of Lemma 6.8. $\square$

In a proof we are about to give, we shall have obtained t and t' such that $t \Rightarrow t'$, and we shall say that we assume the atoms in t' are '**sufficiently fresh for *blah***'. We mean that we choose a $\Rightarrow$ rewrite such that atoms in t' are chosen fresh enough not to clash with certain background assumptions in that proof.

LEMMA 6.12. *If $\Delta \vdash s \to t$ then $\Delta \vdash s \Rightarrow t$, and if $\Delta \vdash s \Rightarrow t$ then $\Delta \vdash s \to^* t$. As a corollary, $\Delta \vdash s \Rightarrow t$ if and only if $\Delta \vdash s \to^* t$.*

PROOF. By induction on the position of the rewrite, and by induction on the derivation of $\Rightarrow$. $\square$

LEMMA 6.13. *If the system is uniform and orthogonal then $\Rightarrow$ is strongly confluent: if $\Delta \vdash s \Rightarrow t$ and $\Delta \vdash s \Rightarrow t'$, then there exists some t'' such that $\Delta \vdash t \Rightarrow t''$ and $\Delta \vdash t' \Rightarrow t''$. Hence $\Rightarrow$ is confluent.*

PROOF. By induction on the derivation of $\Delta \vdash s \Rightarrow t$. We consider one case. Suppose the derivation ends in (tup). By the syntax-driven nature of deduction there are three possibilities for the last rule in the derivation of $\Delta \vdash s \Rightarrow t'$: (tup), (tup'), and (refl):

(1) If $\Delta \vdash s \Rightarrow t'$ has a derivation ending in (tup) then the inductive hypothesis for $\Delta \vdash s_i \Rightarrow t_i$ and $\Delta \vdash s_i \Rightarrow t'_i$ give us t''_i such that $\Delta \vdash t_i \Rightarrow t''_i$ and $\Delta \vdash t'_i \Rightarrow t''_i$. We use (tup) and are done.

(2) If $\Delta \vdash s \Rightarrow t'$ has a derivation ending in (tup') using $R \equiv \nabla \vdash l \to r$, that is $\Delta \vdash s \Rightarrow (t'_1, \dots, t'_n)$ and $\Delta \vdash (t'_1, \dots, t'_n) \stackrel{R_\epsilon}{\to} \stackrel{F}{\to}^* t'$, then θ exits such that

$$\Delta \vdash \ \nabla\theta, \quad (t'_1, \dots, t'_n) \approx_\alpha l\theta, \quad r\theta \approx_\alpha r' \stackrel{F}{\to}^* t'.$$

We now proceed as illustrated and explained below:

$$\begin{array}{ccccccc}
(s_1, \dots, s_n) & \Longrightarrow & (t'_1, \dots, t'_n) & \approx_\alpha & l\theta \xrightarrow{R_\epsilon} r' & \xrightarrow{F}{}^* & t' \\
\Downarrow & & \Downarrow & & & & \Downarrow \\
(t_1, \dots, t_n) & \Longrightarrow & (t''_1, \dots, t''_n) & \approx_\alpha & l\theta' & \xrightarrow{\qquad R_\epsilon \qquad} & r\theta'
\end{array}$$

We apply the inductive hypothesis to close $\Delta \vdash t_i, t'_i \Rightarrow t''_i$, using Lemma 6.11 ($\stackrel{F}{\to}^* \Rightarrow \stackrel{F}{\to}^* \subseteq \Rightarrow$) to ensure the tags in t''_i are sufficiently fresh that Lemma 6.12 ($\to^* = \Rightarrow$) and Lemma 6.3 can be used to deduce of t''_i all freshness and locality assumptions deducible of t'_i.

Since rules are left-linear R still applies: $\Delta \vdash (t''_1, \dots, t''_n) \stackrel{R_\epsilon}{\to} r\theta'$ and therefore $\Delta \vdash (t_1, \dots, t_n) \Rightarrow r\theta'$ by (tup') for R (for some substitution θ'). Finally, we use Lemma 6.8 (F is reversible), Lemma 6.4, orthogonality and Lemma 6.11 to close with a rewrite $t' \Rightarrow r\theta'$.

(3) If $\Delta \vdash s \Rightarrow t'$ is derived using (refl) then by Lemma 6.8 and Lemma 6.11 we get $\Delta \vdash t' \Rightarrow t$, and $\Delta \vdash t \Rightarrow t$ is derivable using (refl).

The other cases are similar; the case of (new) and (new') uses Lemma 6.5. $\square$

We now come back to our theorem:

PROOF. If the uniform rewrite system has only left-linear rules and only trivial critical pairs, then $\Rightarrow$ is confluent by Lemma 6.13. Since $\to^* \subseteq \Rightarrow$ and $\to \subseteq \to^*$ by Lemma 6.12, $\to$ is confluent. $\square$

7. EXTENSIONS: CLOSED, AND IS-IN

We will now show that, thanks to the use of contexts, the framework of nominal rewriting can be easily adapted to express strategies of reduction. As an example, we will show how to define the system λ_{ca} of closed reduction for the λ-calculus [8]. λ_{ca}-terms are λ-terms with explicit constructs for substitutions, copying and erasing. Reduction on λ_{ca} is defined in [8] using a set of conditional rule schemes, shown in Table 1, where x, y, z denote variables, and t, u, v denote terms.

Table 1: λ_{ca}-reduction

Name	Reduction			Condition
Beta	$(\lambda x.t)v$	$\to_{\mathsf{ca}}$	$t[v/x]$	$FV(v) = \emptyset$
Var	$x[v/x]$	$\to_{\mathsf{ca}}$	v	
App1	$(tu)[v/x]$	$\to_{\mathsf{ca}}$	$(t[v/x])u$	$x \in FV(t)$
App2	$(tu)[v/x]$	$\to_{\mathsf{ca}}$	$t(u[v/x])$	$x \in FV(u)$
Lam	$(\lambda y.t)[v/x]$	$\to_{\mathsf{ca}}$	$\lambda y.t[v/x]$	
Copy1	$(\delta_x^{y,z}.t)[v/x]$	$\to_{\mathsf{ca}}$	$t[v/y][v/z]$	
Copy2	$(\delta_{x'}^{y,z}.t)[v/x]$	$\to_{\mathsf{ca}}$	$\delta_{x'}^{y,z}.t[v/x]$	
Erase1	$(\epsilon_x.t)[v/x]$	$\to_{\mathsf{ca}}$	t	
Erase2	$(\epsilon_{x'}.t)[v/x]$	$\to_{\mathsf{ca}}$	$\epsilon_{x'}.t[v/x]$	

We can formally define λ_{ca} using a nominal rewriting system, where we add two new kinds of constraints: $\bullet t$ (read t **is closed**), with the intended meaning "$a\#t$ for every atom a", and $a \in t$ (read a **is unabstracted in** t), the negation of $a\#t$.

We extend the deduction and simplification rules from section 3 respectively with:

$$\frac{(\Delta \vdash a\#t)_{a \in S}}{\Delta \vdash \bullet t}\,(\bullet R) \quad A(t, \Delta) \subsetneq S$$

$$\bullet t, Pr \Longrightarrow \{a\#t\}_{a \in Pr, t}, a'\#t, Pr$$

Here S is any set of atoms strictly containing the atoms in t and Δ, and $a' \notin A(Pr, t)$. In effect we need $A(t, \Delta)$ *and* one fresh atom; if $\Delta \vdash a\#t$ for $a \in A(t, \Delta)$ a renaming argument gives $\Delta \vdash b\#t$ for all other $b \notin A(t, \Delta)$. This is reflected in the simplification rule, which is more algorithmic and chooses one fresh atom.

$\bullet t$ is intuitively $\forall a.\ a\#t$. The rule for closure $\bullet t$ is slightly different from the usual predicate logic rule for $\forall$ because atoms behave here as constants and not variables. With that in mind the definitions are quite natural.

We can extend the deductions with rules including

$$\frac{a \in t_i}{a \in (t_1, \dots, t_n)} \qquad \frac{}{a \in a} \qquad \frac{a \in t}{a \in \text{И}a.t}$$

and similarly extend the simplification rules. We can extend contexts with these new constraints and use them in ∇s of rewrite rules $\nabla \vdash l \to r$ to control triggering.

A closed reduction strategy can be specified, this time as a finite nominal rewrite system (we only show rules *Beta*, *App*$_1$ and *App*$_2$):

$$\begin{array}{llll}
\textit{Beta} & \bullet V \vdash (\lambda[x]T)V & \to & T[V/x] \\
\textit{App1} & x \in T \vdash (TU)[V/x] & \to & (T[V/x])U \\
\textit{App2} & x \in U \vdash (TU)[V/x] & \to & T(U[V/x])
\end{array}$$

8. CONCLUSIONS AND FUTURE WORK

The technical foundations of this work are derived from work on nominal logic [20], nominal unification [23] and nominal rewriting [6]. Here we extended nominal terms and the unification algorithm with a new construct to model restriction of names. We also showed that the nominal rewriting framework can be easily adapted to model reduction strategies: the context used to trigger rules allows us to express constraints in the application of the rules.

There is interest in extending Structural Operational Semantics with abstraction and name-generation and Nominal Rewriting could provide (yet another!) forum for it. The authors have considered efficient implementation of N; there is no space here (we treated orthogonality instead) but the issue needs to be addressed. We have shown how rewriting can be extended with contexts (in the case of this paper, $\#$, $\in$, and $\bullet$; other constraints can be added to model other domains) and annotations on terms ($\mathsf{N}A$), and we have shown how to retain classic properties of first-order rewriting (in this case, critical pairs and orthogonal rewriting results). We can encode useful structure in rules (name abstraction and generation, and rewrite strategies). It should be possible to consider different extensions, using the same idea of rewriting-in-context.

Abstraction vs. N.

Abstraction and name-generation have different properties. The theory of $\approx_\alpha$ for abstraction is generated by $a\#[a]X$ and $a\#X, b\#X \vdash X \approx_\alpha (a\ b)\cdot X$. The theory of $\approx_\alpha$ for generation is generated by $a@\mathsf{N}a.X$ and $a@X \vdash X \approx_\alpha \mathsf{N}a.X$. N and abstraction are independent in that neither depends on the other for its construction. They coexist in our system with quite subtle interactions. The relevant results are: $\#$ implies $@$ by Lemma 3.1, which comes down to a deduction rule ($\#@$). $@$ implies $\#$ indirectly through Lemma 6.3, which comes down to a rewrite rule F.

Indeed equality itself is rather subtle; it is constructed in layers starting with $\equiv$ (syntactic identity, an intrinsic property of terms), then $\approx_\alpha$ defined using a context and handling abstraction $[a]s$, and finally rule F which is built into the rewriting and deals with N (as we see from Lemma 6.3 and the final parts of §6.2 and §6.3). Finding these different layers, and understanding their interactions, were the principal technical difficulties in this work.

Can N model abstraction, or vice versa? Abstraction cannot directly model N, because N does not abstract and of course abstraction does. The converse does not hold, since N with α-renaming rules could model the abstraction part of abstraction, but we still have $a@X \vdash X \approx_\alpha \mathsf{N}a.X$, which abstraction should not satisfy.

Recall our reaction systems for the π-calculus and the ν-calculus from §5.2. We used abstraction to model the π-calculus term $\nu a.\, P$ as $\nu[a]P$ because the a is normally considered α-convertible in the π-calculus literature. But in fact we could have used N directly and saved ourselves the rewrite $\nu[a]X \to \mathsf{N}a.X$:

$$\overline{a}b \mid a[c].Y \to Y\{c{\mapsto}b\} \quad a\#P \vdash P|(\mathsf{N}a.Q) \to \mathsf{N}a.(P|Q)$$
$$!X \to !X \mid X \quad a\#Q \vdash (\mathsf{N}a.P)|Q \to \mathsf{N}a.(P|Q).$$

We *cannot* model λ-abstraction as $\lambda\mathsf{N}a.t$, since $\lambda\mathsf{N}a.b \approx_\alpha \lambda b$ is provable. We can model $\nu a.\, t$ in the ν-calculus with N as $\nu\mathsf{N}a.t$ (hence using only one rule: $(\lambda[a]X)Y \to X\{a{\mapsto}Y\}$).

Acknowledgements: We thank James Cheney, Nobuko Yoshida, and the referees, for useful comments.

9. REFERENCES

[1] F. Baader and T. Nipkow, *Term rewriting and all that*, Cambridge University Press, 1998.

[2] H. P. Barendregt, *The Lambda Calculus: its Syntax and Semantics (revised ed.)*, Studies in Logic and the Foundations of Mathematics, Vol. 103, North-Holland, 1984.

[3] H. Cirstea and C. Kirchner, *The Rewriting Calculus - Part I*, Logic Journal of the Interest Group in Pure and Applied Logics **9** (2001), 363–399.

[4] H. Cirstea and C. Kirchner, *The Rewriting Calculus - Part II*, Logic Journal of the Interest Group in Pure and Applied Logics **9** (2001), 401–434.

[5] N. Dershowitz and J.-P. Jouannaud, *Rewrite Systems*, Handbook of Theoretical Computer Science: Formal Methods and Semantics, vol. B, 1989.

[6] M. Fernández, M. J. Gabbay and I. Mackie,*Nominal Rewriting Systems*, ACM Symposium on Principles and Practice of Declarative Programming (PPDP'04), ACM Press, 2004.

[7] M. Fernández and J-P. Jouannaud. *Modular Termination of Term Rewriting Systems Revisited.* ADT'94, LNCS 906, Springer-Verlag, 1994.

[8] M. Fernández, I. Mackie and F-R. Sinot, *Closed Reduction: Explicit Substitutions without alpha-conversion*, Mathematical Structures in Computer Science 15:2, 2005.

[9] M. J. Gabbay and J. Cheney, *A Sequent Calculus for Nominal Logic*, Proc. 19th IEEE Symposium on Logic in Computer Science (LICS'04), pages 139–148, 2004.

[10] M. J. Gabbay and A. M. Pitts, *A New Approach to Abstract Syntax with Variable Binding.* Formal Aspects of Computing, vol. 13, pp. 341–363, 2002.

[11] Z. Khasidashvili, *Expression reduction systems*, Proceedings of I.Vekua Institute of Applied Mathematics (Tbisili), vol. 36, 1990, pp. 200–220.

[12] J.-W. Klop, *Term Rewriting Systems*, Handbook of Logic in Computer Science, Oxford University Press, 1992.

[13] J.-W. Klop, V. van Oostrom, and F. van Raamsdonk, *Combinatory reduction systems, introduction and survey*, Theoretical Computer Science **121** (1993), 279–308.

[14] J-W. Klop, *Combinatory reduction systems*, Mathematical Centre Tracts, vol. 127, Amsterdam, 1980.

[15] R. Mayr and T. Nipkow, *Higher-order rewrite systems and their confluence*, TCS **192** (1998), 3–29.

[16] D. Miller and A. Tiu, *A proof theory for generic judgments: An extended abstract*, LICS 2003, 118–127.

[17] Milner, Parrow, and Walker, *A Calculus of Mobile Processes, II*, Inf. and Comp. 100:1 1992, 41–77.

[18] A. Momigliano and F. Pfenning, *Higher-order Pattern Complement and the Strict λ-calculus*, ACM Trans. Comput. Logic 4:4, 493–529, ACM Press, 2003.

[19] M.H.A. Newman, *On theories with a combinatorial definition of equivalence*, Annals of Mathematics **43** (1942), no. 2, 223–243.

[20] A. M. Pitts, *Nominal logic, a first order theory of names and binding*, Information and Computation **186** (2003), 165–193.

[21] A. M. Pitts and I. D. B. Stark, *Observable Properties of Higher Order Functions That Dynamically Create Local Names, or: What's new?*. LNCS 711, 1993.

[22] A. M. Pitts and M. J. Gabbay, *A metalanguage for programming with bound names modulo renaming*, MPC2000, LNCS 1837, 2000.

[23] C. Urban, A. M. Pitts, and M. J. Gabbay, *Nominal unification*, CSL'03 & KGC. LNCS 2803, 2003.