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 explicitly named (rather than being nameless, as for de Bruijn indices) yet rewriting respects α-conversion and can be directly implemented, thanks to the use of freshness contexts. In this paper we study two extensions to nominal rewriting. First we introduce a И quantifier for modelling name generation. 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

Term rewriting systems (TRS) specify and reason about computation by working with trees labelled by variable and function symbols. Standard TRSs are first-order, but effort has been devoted to systems manipulating higher-order functions, where variables can be free or bound. Examples of higher-order rewriting formalisms combining first-order rewriting with a notion of bound variable are: Combinatory Reduction Systems (CRS) [24], Higher-order Rewrite Systems (HRS) [27], Expression Reduction Systems (ERS) [22], and the rewriting calculus [10, 11]. The λ-calculus can be seen as a higher-order rewrite system with one binder called λ-abstraction.

Binders lead to α-conversion. First-order substitutions (i.e. replacements) may capture variables, so substitution in a higher-order system has to be carefully defined using renamings (α-conversion). In the higher-order formalisms mentioned above substitution is a meta-operation which relies on an implicit notion of α-conversion: terms are defined modulo renamings of bound variables. Several notions of explicit substitutions and explicit α-conversion have been defined for the λ-calculus (e.g. [1, 26, 12]) and more generally for higher-order rewrite systems (e.g. [30, 6]) with the aim of specifying the higher-order notion of substitution as a set of first-order rewrite rules. In most of these systems variable names are replaced by de Bruijn indices to make easier the explicitation of α-conversion, at the expense of readability.

Nominal rewriting is a new formalism for rewriting with bound variables, introduced in [14]. In nominal rewriting systems (NRSs) variables are named (whence 'nominal'), substitution is first-order, and we deal with α-conversion by using an auxiliary **freshness** relation $a\#t$ between variables and terms (we say "a is fresh for the term t"). This freshness relation was introduced in [31, 18], and used to define a metalanguage for functional programming in [33], and to study unification problems in [36].

Nominal terms are first-order terms with built-in α-conversion. Our metalanguage does not include metasubstitutions and β-reductions (in contrast to notions of higher-order rewriting): in this sense, nominal rewriting systems are a middle ground between first-order and higher-order rewriting. The usual notion of higher-order substitution can be easily specified since α-conversion *is* in the meta-language.

Following [31, 18], we call the names that can be bound **atoms** and reserve the word **variable** for the identifiers that cannot be bound (known as variables and metavariables respectively in CRSs). In nominal terms the dependencies between variables and names are implicit, as in informal presentations of higher-order reductions. For example, λ-calculus β- and η-rules are written

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

Rewriting on nominal terms uses **nominal matching** [14, 36, 37], which does not need metasubstitutions and β-reduction but it does take into account freshness contexts. Selecting a nominal rewrite rule that matches a given term is an NP-complete problem in general [9]; however, by restricting to **closed** rules we can avoid the exponential cost: nominal matching is polynomial in this case [14]. Closed nominal rewriting turns out to be sufficiently close to first-order rewriting to share many of its properties, including a critical pair lemma [14], and a confluence theorem for orthogonal (i.e. left-linear and non-overlapping) systems which we prove in this paper. It is also expressive (see [14] for a translation of Klop's Combinatory Reduction Systems) but has some limitations: rewrite rules have no mechanism for locally generating a fresh atom. This occurs in nature. For example in a rewrite system for the π-calculus [28] we might wish a reaction rule which we might write as $\nu[n]X \to \text{И}n.X$, where И indicates 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.

In this paper we study two extensions to nominal rewriting:

(1) We consider the **name generation** construct И and make corresponding extensions to the freshness contexts to describe the **scope** of the names thus generated. Although И can be seen as a quantifier modelling locality of names, it is not a binder: we will show that abstracted names and local names have different behaviour.

(2) We extend just the contexts with new predicates specifying constraints for atoms in terms, for instance closedness. In this way strategies of reduction such as closed reduction in the λ-calculus [16] may be concisely specified. This extension has applications in programming language design and implementation (see for instance [16]).

We study confluence properties of extended nominal rewriting: we give a critical pair lemma for extended NRSs and we prove that orthogonal systems are confluent.

In summary the main contributions of this paper are:

1. We formalise a notion of name generation and locality on nominal terms, with an associated notion of rewriting, with applications to distributed and higher-order programming.

2. We define two new predicates: a scope relation and a closedness relation, which can be used as constraints together with the freshness relation in contexts to specify strategies of reduction in rewrite rules.

3. A Critical Pair Lemma which ensures that extended nominal rewriting rules which do not introduce critical pairs are locally confluent.

4. A Confluence Theorem for orthogonal systems.

Related Work.

NRSs are related to algebraic λ-calculi [7, 8, 21, 3] where the β-reduction rule of the λ-calculus is combined with a set of term rewriting rules using standard first-order matching. Nominal rules are more general than these, since abstractions can be used in patterns, in the same way as in higher-order rewriting systems (e.g. CRSs [24], HRSs [27], ERSs [22], HORSs [34], and the ρ-calculus [10, 11]).

In contrast with higher-order rewriting systems, nominal rewriting does not need higher-order matching and higher-order substitutions. Instead, it relies on an extension of first-order matching that takes care of α-conversion without introducing β-reductions.

Although NRSs were not designed as explicit substitution systems, they are at an intermediate level between standard higher-order rewriting systems and their explicit substitution versions (e.g. [30, 6]), which implement in a first-order setting the substitution operation together with α-conversions using de Bruijn indices. Nominal rewriting is also related to Hamana's Binding Term Rewriting Systems (BTRS) [19]. The main difference is that BTRSs use a containment relation that indicates which free atoms occur in a term (as opposed to a freshness relation which indicates that an atom does not occur free in a term).

The name generation operator И of extended nominal rewriting systems is related to the ν operator of Pitts and Stark's ν-calculus [32] and ν the π-calculus restriction operator [28]. Apart from obvious differences between a simply typed λ-calculus, a process calculus, and a rewriting formalism, ν and И differ in their treatment of generated names: ν is a binder, whereas И is a scope construct.

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 [14] 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 will represent meta-level unknowns. Fix a distinct countably infinite set $\mathcal{A}$ of **atoms** $a, b, c, f, g, h, \ldots$. These will 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.

A **swapping** is a pair of atoms, which we write $(a\ b)$. **Permutations** π are 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 $\pi\cdot X$. Formally, swappings and moderated variables are just pairs, and permutations are just lists of swappings.

Extended Nominal Terms, also called **tagged terms**

or just **terms** for short, are generated by the grammar

$$t ::= \text{И}A.a \mid \text{И}A.\pi{\cdot}X \mid \text{И}A.(t_1,\ldots,t_n) \mid \text{И}A.[a]t \mid \text{И}A.(ft),$$

where we write A for a finite set of atoms which may be empty. We may write $\text{И}\emptyset.blah$ more succintly as $blah$, 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. In the clause for function applications, f may be applied to the empty tuple in which case we may write $f()$ as just f.

Note that an extended nominal term is a tree with nodes annotated by function symbols or (moderated) variables, so, syntax in the accepted sense — later we call it the term's **skeleton** — but also annotated at every node with a finite sets of atoms. We shall call these sets **tags**. A term such that every tag is empty is in essence ordinary syntax and is isomorphic to a(n unextended) nominal term [14].

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 $\text{И}\{a\}.()$ 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, $[a]a \not\equiv [b]b$. In nominal techniques, 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 $\text{И}\{a\}.t$ is intended to represent t with a local name it is calling a. We call occurrences of a **scoped** or **local**, and unscoped occurrences **unscoped** or **global**. As with abstraction, we do not work up to α-conversion of tags.

Abstraction is inherited from previous work [36, 14], tags are new to this paper. We give some examples:

- $\text{И}\{a\}.a$ is a scoped copy of a. In future, we drop the set curly brackets and write $\text{И}a.a$.
- $\text{И}b.b$ is a local name called b. Note that $\text{И}a.a \not\equiv \text{И}b.b$.
- $\text{И}a.b$ is b with a local name a (which is not used, the reader might like to consider it '**garbage**'). $\text{И}a.b$ is not syntactically identical to $\text{И}c.b$, $\text{И}b.b$, or b.
- $\text{И}a,b.a$ generates a and b and discards b. Because the tags are sets, $\text{И}a,b.a \equiv \text{И}b,a.a$. However, $\text{И}a,b.a \not\equiv \text{И}a,b.b$.
- $\text{И}a.(\text{И}\emptyset.a, \text{И}\emptyset.a)$ generates a and makes the pair (a,a).
- $(\text{И}a.a, \text{И}a.a)$ generates two local names which are both (by coincidence) 'a'.
- $\text{И}a.(a, \text{И}a.a)$ 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 its own local scope. $\text{И}a.(a,\text{И}a.a) \not\equiv (\text{И}a.a, \text{И}a.a)$.
- Using the ML signature mentioned above, we can write $(\mathtt{lam}[a]\text{И}b.(\mathtt{app}(\mathtt{var}(a), \mathtt{app}(\mathtt{var}(b), X))))$ which we will abbreviate $\lambda[a]\text{И}b.a(bX)$. 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:

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

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:

$$(a\ b){\cdot}\text{И}A.n \equiv \text{И}(a\ b)(A).(a\ b)(n)$$
$$(a\ b){\cdot}\text{И}A.(ft) \equiv \text{И}(a\ b)(A).(f(a\ b){\cdot}t)$$
$$(a\ b){\cdot}\text{И}A.(t_1,\ldots,t_n) \equiv \text{И}(a\ b)(A).((a\ b){\cdot}t_1,\ldots,(a\ b){\cdot}t_n)$$
$$(a\ b){\cdot}\text{И}A.[n]t \equiv \text{И}(a\ b)(A).[(a\ b)(n)](a\ b){\cdot}t$$
$$(a\ b){\cdot}\text{И}A.\pi{\cdot}X \equiv \text{И}(a\ b)(A).((a\ b)\cdot\pi){\cdot}X.$$

Here we have applied swappings to atoms and sets of atoms $(a\ b)(n)$ and $(a\ b)(A)$. The action on sets is pointwise, and the action on atoms satisfies:

$$(a\ b)(a) \equiv b \qquad (a\ b)(b) \equiv a \quad \text{and} \quad (a\ b)(c) \equiv c\ (c \not\equiv a, b).$$

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

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 [13, 23]; see also [2]. '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 **scoping** (or **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).

In the following rules, and consistent with notation used above, we write $a\#\text{И}\emptyset.\mathbf{Id}{\cdot}X$ as $a\#X$, $a@\text{И}\emptyset.\mathbf{Id}{\cdot}X$ as $a@X$, $\text{И}\emptyset.\mathbf{Id}{\cdot}X \approx_\alpha t$ as $X \approx_\alpha t$, and $s \approx_\alpha \text{И}\emptyset.\mathbf{Id}{\cdot}Y$ as $s \approx_\alpha Y$.

$a\#s$ is specified by **deduction rules** as follows:

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

$$\frac{}{a\#\text{И}A.[a]s} \qquad \frac{a\#s}{a\#\text{И}A.[b]s} \qquad \frac{\pi^{-1}(a)\#X}{a\#\text{И}A.\pi{\cdot}X}$$

This definition is like the one in [14] except that terms have tags added; to calculate $\#$ we simply throw them out.

$a@s$ is specified by **deduction rules** as follows:

$$\frac{a\#X}{a@X}\,(\#@) \qquad \frac{a@\text{И}A.a}{P}\,(@\bot)\ a\notin A$$

$$\frac{}{a@\text{И}A.s}\ a\in A \qquad \frac{a@s_1 \ \cdots\ a@s_n}{a@\text{И}A.(s_1,\ldots,s_n)}\ a\notin A$$

$$\frac{a@s}{a@\text{И}A.fs}\ a\notin A \qquad \frac{a@s}{a@\text{И}A.[b]s}\ a\notin A \qquad \frac{}{a@\text{И}A.b}\ a\notin A$$

$$\frac{}{a@\text{И}A.[a]s}\ a\notin A \qquad \frac{\pi^{-1}(a)@X}{a@\text{И}A.\pi{\cdot}X}\ a\notin A$$

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

Finally we define $\approx_\alpha$ inductively as follows, where in all the rules except the last s and t are terms with an empty tag at the root (i.e. $\text{И}\emptyset.s$, $\text{И}\emptyset.t$).

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

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

$$\frac{\text{И}\emptyset.s\approx_\alpha\text{И}\emptyset.t \qquad B\backslash A@\text{И}\emptyset.s \qquad A\backslash B@\text{И}\emptyset.t}{\text{И}A.s\approx_\alpha\text{И}B.t}\ A\cup B\neq\emptyset$$

We have used the **difference set** of two permutations:

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

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 $[a]a \approx_\alpha [b]b$. Other examples of α-equivalence are: $\text{И}a.b \approx_\alpha b$, 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 $a\#\text{И}A.a$, $a\#X$, $a@X$, and $a@\text{И}A.a$ with $a\notin A$, are **reduced**. We write Δ,∇,Γ for sets of reduced constraints, we may call them **contexts**. If there are no constraints of the form $a\#\text{И}A.a$, and $a@\text{И}A.a$ with $a\notin A$ 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 $(\#\text{И})$ to deduce $a@X$ from some $a\#X\in\nabla$. $\square$

Lemma 3.2. *Suppose ∇ is a reduced context.*

1. *$\nabla\vdash a\#\pi{\cdot}t$ if and only if $\nabla\vdash\pi^{-1}(a)\#t$.*
2. *$\nabla\vdash a@\pi{\cdot}t$ if and only if $\nabla\vdash\pi^{-1}(a)@t$.*
3. *$\nabla\vdash a\#\text{И}A.s$ if and only if $\nabla\vdash a\#\text{И}B.s$.*
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 $\nabla\vdash a@t'$.*

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 $\text{И}\emptyset.[a]s\approx_\alpha\text{И}\emptyset.[b]t$, and the third part for the case of $\text{И}A.s\approx_\alpha\text{И}B.t$.

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

Suppose $\nabla\vdash x@\text{И}A.t$ and $\nabla\vdash\text{И}A.t\approx_\alpha\text{И}A'.t'$ are derivable where at least one of A and A' is nonempty. Then $\nabla\vdash c@t'$ for every $c\in A'\setminus A$, and $\nabla\vdash c@t$ for every $c\in A\setminus A'$, and $\nabla\vdash\text{И}\emptyset.t\approx_\alpha\text{И}\emptyset.t'$. There are now three cases:

- If $x\in A'$ then $\nabla\vdash x@\text{И}A'.t'$ and we are done.
- If $x\in A$ and $x\notin A'$ then $\nabla\vdash x@t'$ and so $\nabla\vdash x@\text{И}A'.t'$ and we are done.
- If $x\notin A$ and $x\notin A'$ then $\nabla\vdash x@t$ and $\nabla\vdash\text{И}\emptyset.t\approx_\alpha\text{И}\emptyset.t'$ and we use the inductive hypothesis.

Suppose $\nabla\vdash x@\text{И}\emptyset.[a]t$ and $\nabla\vdash\text{И}\emptyset.[a]t\approx_\alpha\text{И}\emptyset.[a']t'$. 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 $\nabla \vdash x@\text{И}\emptyset.(a\ a')\cdot t'$, the second part of this lemma to deduce $\nabla \vdash x@t'$, and finally extend to a derivation of $\nabla \vdash x@\text{И}\emptyset.[a']t'$ as required.

□

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. For transitivity, we use induction on derivations and the previous lemma for the case of $\text{И}A.[x]s \approx_\alpha \text{И}A'.[x']s' \approx_\alpha \text{И}A''.[x'']s''$. The final part is also by induction; only the case of $[a]s \approx_\alpha [b]t$ is in any way non-trivial, we use the previous lemma. □

COROLLARY 3.4.
- *$\Delta \vdash s \approx_\alpha \pi^{-1}\cdot\pi\cdot s'$ if and only if $\Delta \vdash s \approx_\alpha s'$.*
- *$\Delta \vdash s \approx_\alpha \pi\cdot s'$ if and only if $\Delta \vdash \pi^{-1}\cdot s \approx_\alpha s'$.*
- *$\Delta \vdash ds(\pi, \pi')\#s$ then $\Delta \vdash \pi\cdot s \approx_\alpha \pi'\cdot s$.*
- *$\Delta \vdash A \setminus A'@s$ and $\Delta \vdash A' \setminus A@s$ then $\Delta \vdash \text{И}A.s \approx_\alpha \text{И}A'.s$.*
- *For fixed Δ, $\approx_\alpha$ is a congruence. Thus, if $\Delta \vdash t \approx_\alpha t'$ then $\Delta \vdash s[t]_p \approx_\alpha s[t']_p$.*

PROOF. The first four parts are by routine inductions using the previous results. The last part is by induction on the syntax of s and uses the previous cases. □

3.2 An algorithmic presentation

We specify **simplification rules** on problems by:

$$\begin{array}{rcl}
a\#\text{И}A.b, Pr & \Longrightarrow & Pr \\
a\#\text{И}A.fs, Pr & \Longrightarrow & a\#s, Pr \\
a\#\text{И}A.(l_1,\ldots,l_n), Pr & \Longrightarrow & a\#l_1,\ldots,a\#l_n, Pr \\
a\#\text{И}A.[b]s, Pr & \Longrightarrow & a\#s, Pr \\
a\#\text{И}A.[a]s, Pr & \Longrightarrow & Pr \\
a\#\text{И}A.\pi\cdot X, Pr & \Longrightarrow & \pi^{-1}(a)\#X, Pr
\end{array}$$

$$\begin{array}{rcll}
a@\text{И}A.s, Pr & \Longrightarrow & Pr & a \in A \\
a@\text{И}A.b, Pr & \Longrightarrow & Pr & a \notin A \\
a@\text{И}A.fs, Pr & \Longrightarrow & a@s, Pr & a \notin A \\
a@\text{И}A.(l_1,\ldots,l_n), Pr & \Longrightarrow & a@l_1,\ldots,a@l_n, Pr & a \notin A \\
a@\text{И}A.[b]s, Pr & \Longrightarrow & a@s, Pr & a \notin A \\
a@\text{И}A.[a]s, Pr & \Longrightarrow & Pr & a \notin A \\
a@\text{И}A.\pi\cdot X, Pr & \Longrightarrow & \pi^{-1}(a)@X, Pr & a \notin A
\end{array}$$

$$\begin{array}{rcll}
(l_1,\ldots,l_n) & \Longrightarrow & l_1 \approx_\alpha s_1,\ldots, & \\
\approx_\alpha (s_1,\ldots,s_n), Pr & & \quad l_n \approx_\alpha s_n, Pr & \\
fl \approx_\alpha fs, Pr & \Longrightarrow & l \approx_\alpha s, Pr & \\
[a]l \approx_\alpha [a]s, Pr & \Longrightarrow & l \approx_\alpha s, Pr & \\
[b]l \approx_\alpha [a]s, Pr & \Longrightarrow & (a\ b)\cdot l \approx_\alpha s, a\#l, Pr & \\
a \approx_\alpha a, Pr & \Longrightarrow & Pr & \\
\pi\cdot X \approx_\alpha \pi'\cdot X, Pr & \Longrightarrow & ds(\pi,\pi')\#X, Pr & \\
\text{И}A.s \approx_\alpha \text{И}B.t, Pr & \Longrightarrow & B\backslash A@\text{И}\emptyset.s,\ A\backslash B@\text{И}\emptyset.t, & \\
& & \text{И}\emptyset.s \approx_\alpha \text{И}\emptyset.t,\ Pr & A \cup B \neq \emptyset
\end{array}$$

Here, on the left commas indicate disjoint set union, on the right they indicate possibly non-disjoint set union.

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 last rule reduces the problem $\{\text{И}a.a \approx_\alpha \text{И}\emptyset.a,\ \text{И}\emptyset.a \approx_\alpha \text{И}\emptyset.a\}$ to $\{a@\text{И}\emptyset.a,\ \text{И}\emptyset.a \approx_\alpha \text{И}\emptyset.a\}$. 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 $(\#\text{И})$, and also rules corresponding to $(\#\bot)$ and $(@\bot)$. If we didn't, the important lemmas below would not hold.

LEMMA 3.5. *The simplification rules above are confluent and strongly normalising (reduction order does not matter).*

PROOF. By Newman's Lemma [29] we need only show termination, because there are no critical pairs. The rules form a hierarchical system in the sense of [15], 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. □

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 $\text{И}\emptyset.s \approx_\alpha \text{И}\emptyset.t$ is **clashing** when the terms s and t have different term constructors at the root (e.g. $[a]u \approx_\alpha (v,w)$, $X \approx_\alpha ft$, $a \approx_\alpha X$, 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 3.6 (NORMAL FORMS). *$\langle a\#s\rangle_{nf}$ consists of reduced freshness constraints and $\langle a@s\rangle_{nf}$ consists of reduced scoping constraints. $\langle s \approx_\alpha t\rangle_{nf}$ may contain any mixture of reduced freshness constraints, reduced scoping constraints, and clashing equalities.*

PROOF. Any non-reduced freshness or scoping 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. □

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 3.7. *Assume $Pr \Longrightarrow Pr'$. Then*

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

PROOF. By inspection of the rules. □

THEOREM 3.8. *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 3.7, $\Delta \vdash Pr$ if and only if $\Delta \vdash Eq$. The result follows from the fact that if $s \approx_\alpha t$ is clashing then there is no consistent Δ such that $\Delta \vdash s \approx_\alpha t$.

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

COROLLARY 3.9. *Let* Γ *be a consistent context.*

$$\Gamma \vdash Pr \textit{ if and only if } \langle Pr\rangle_{nf} \equiv \Delta \textit{ and } \Gamma \vdash \Delta.$$

PROOF. Consequence of Theorem 3.8 and Lemma 3.7. $\square$

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.

Write $\Delta \vdash \sigma \approx_\alpha \sigma'$ when $\Delta \vdash X\sigma \approx_\alpha X\sigma'$ for all X.

Write $\mathbf{Rt}(s)$ for the tag associated to the root of s (so $\mathbf{Rt}(\mathsf{V\!\!\!/}a, b.(\mathsf{V\!\!\!/}c, a.U, V)) = \{a, b\}$).

Write $(\Gamma, \sigma) \leq (\Gamma', \sigma')$ when there exists a substitution σ'' such that $\Gamma' \vdash \Gamma\sigma''$, $\Gamma' \vdash \sigma \circ \sigma'' \approx_\alpha \sigma'$ and $\mathbf{Rt}(X\sigma) \subseteq \mathbf{Rt}(X\sigma')$ for every X. We see that this defines a partial order on solutions (sometimes called instantiation ordering). Say a solution to a problem is **principal** when it is smaller than or equal to all other solutions to that problem. A solution (Γ, σ) is idempotent when $\Gamma \vdash \sigma \circ \sigma \approx_\alpha \sigma$.

We will show below that every solvable unification problem has a principal idempotent solution, and give an algorithm to find it.

The simplification rules given in the previous section can be adapted to solve unification problems: we need to add **instantiating** rules, labelled with substitutions.

$$\begin{array}{llll} \mathsf{V\!\!\!/}\emptyset.\pi\cdot Y\ {}_?\!\!\approx_?\ \mathsf{V\!\!\!/}\emptyset.s, Pr & \stackrel{Y\mapsto\pi^{-1}\cdot s}{\Longrightarrow} & Pr[Y\mapsto\pi^{-1}\cdot s] & (Y \notin s) \\ \mathsf{V\!\!\!/}\emptyset.l\ {}_?\!\!\approx_?\ \mathsf{V\!\!\!/}\emptyset.\pi\cdot X, Pr & \stackrel{X\mapsto\pi^{-1}\cdot l}{\Longrightarrow} & Pr[X\mapsto\pi^{-1}\cdot l] & (X \notin l) \end{array}$$

The conditions in the rules are usually called **occurs check**.

Note that we do not intend to *solve* apartness or scope constraints. Indeed, to solve apartness and scope constraints we need instantiating rules for these predicates too. Instead, we will solve the equalities in a problem, and require that the solution satisfy the apartness and scope constraints in the problem. This is what we need in rewriting.

We will use the set of simplification rules given in the previous section (where we replace $\approx_\alpha$ by ${}_?\!\!\approx_?$), together with the instantiation rules, as a unification algorithm.

LEMMA 4.1. *The unification 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 simplifies towards reduced constraints. $\square$

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 using the unification rules, and σ is obtained by concatenating the labels in some reduction sequence from Pr to Pr'. We can characterise the normal form Pr' of Pr under the unification rules:

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 scoping constraints.* $\langle s\ {}_?\!\!\approx_?\ t\rangle_{sol}$ *may contain any mixture of reduced freshness constraints, reduced scoping constraints, and clashing equalities such that if one side is a variable the other side contains this variable.*

PROOF. Similar to Lemma 3.6: Any non-reduced freshness or scoping constraint or non-clashing equality can be simplified as previously. Clashing equalities of the form $\pi\cdot X\ {}_?\!\!\approx_?\ t$ or $t\ {}_?\!\!\approx_?\ \pi\cdot X$ where $X \notin V(t)$ 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. Let* Eq *be a non-empty set of unification constraints in normal form. Then* Eq *has no solution.*

2. Let Δ *be an inconsistent context. Then* Δ *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* $\Delta \vdash (\pi\cdot X)\sigma \approx_\alpha t\sigma$ *then* $\Delta \vdash [X\mapsto\pi^{-1}\cdot t] \circ \sigma \approx_\alpha \sigma$.

Similarly, if $\Delta \vdash s\sigma \approx_\alpha (\pi\cdot Y)\sigma$ *then* $\Delta \vdash [Y\mapsto\pi^{-1}\cdot s] \circ \sigma \approx_\alpha \sigma$.

PROOF. Suppose $\Delta \vdash (\pi\cdot X)\sigma \approx_\alpha t\sigma$ and write σ' for $[X\mapsto\pi^{-1}\cdot t] \circ \sigma$. We simplify $X\sigma'$ up to $\approx_\alpha$ in the context Δ;

$$\begin{aligned} X\sigma' &\approx_\alpha (\pi^{-1}\cdot t)\sigma \approx_\alpha \pi^{-1}\cdot(t\sigma) \\ &\approx_\alpha \pi^{-1}\cdot((\pi\cdot X)\sigma) \approx_\alpha \pi^{-1} \circ \pi\cdot(X\sigma) \approx_\alpha X\sigma. \end{aligned}$$

(We make heavy use of the previous lemma to rearrange the brackets.) The second part is similar. $\square$

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 $\Gamma \vdash \theta \circ \sigma \approx_\alpha \sigma$.*

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

Proof. The first part follows from the previous lemma.
For the second part, suppose $\Gamma \vdash Pr'\sigma$ and suppose $\theta = [X{\mapsto}\pi^{-1}{\cdot}s]$ so that $(\{s \approx_\alpha s\} \cup Pr') = Pr[X{\mapsto}\pi^{-1}s]$. It is easy to verify that $\Gamma \vdash s \approx_\alpha s$, 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 a consistent context and $Eq = \emptyset$.

Proof. By induction on the length of the derivation $Pr \overset{*}{\Longrightarrow} \langle Pr\rangle_{sol}$. 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 can be proved by induction on the derivation $Pr \overset{*}{\Longrightarrow} \langle Pr\rangle_{sol}$. A step $\pi{\cdot}X \;{}_?{\approx}_?\; t, Pr \overset{X\mapsto\pi^{-1}\cdot t}{\Longrightarrow} Pr[X{\mapsto}\pi^{-1}{\cdot}t]$ is replaced by $t \approx_\alpha t, Pr[X{\mapsto}\pi^{-1}{\cdot}t] \overset{*}{\Longrightarrow} Pr[X{\mapsto}\pi^{-1}{\cdot}t]$. $\square$

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 $(\Delta, Id) \in \mathcal{U}(\Delta)$ 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:

- $\Gamma \vdash \sigma{\circ}\sigma' \approx_\alpha \sigma'$, by Lemma 4.6 part 1 (and induction). Note in particular that $\Gamma \vdash \sigma \circ \sigma \approx_\alpha \sigma$, that is, σ is idempotent.

- $\Gamma \vdash Pr\sigma'$ by definition of solution, and $\Gamma \vdash \langle Pr\sigma'\rangle_{nf}$ by Corollary 3.9. 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 3.9.

- Finally, $\mathbf{Rt}(X\sigma) \subseteq \mathbf{Rt}(X\sigma')$ for every X since σ is created by the instantiation rules with empty tags.

Therefore (Δ,σ) is a principal and idempotent solution.

$\square$

5. REWRITING

5.1 Matching problems

Given terms-in-context $\nabla \vdash l$ and $\Delta \vdash s$ such that $V(\nabla, l) \cap V(\Delta, s) = \emptyset$, a **matching problem** between them is the pair $(\nabla \vdash l, \Delta \vdash s)$. We write it $(\nabla \vdash l) \;{}_?{\approx}\; (\Delta \vdash s)$. 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
- $\Delta \vdash \Delta',\ l\theta \approx_\alpha s$ is derivable.
- $\Delta \vdash \nabla\theta$ is derivable.

The first condition ensures that our solution is a matching. The second condition controls when l matches with s. The third condition controls when the conditions ∇ are satisfied. When this third condition is satisfied we say the matching **is triggered**. (Soon, $\nabla \vdash l$ will be the left-hand 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 $(a@X, [X'{\mapsto}X])$:

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

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

See [14] 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 $\Delta \vdash s \overset{R}{\to} t$ 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 [14], 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 rewrites 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 [14].

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. This is because of equivariance: if $a \to a \in \mathcal{R}$ then so is $n \to n$ for any n.

  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 $\Delta \vdash s \xrightarrow{R} t$ then $\Delta \vdash C[s] \xrightarrow{R} C[t]$.*

Proof. By unpacking the definition of rewriting. $\square$

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. We assume $\mathbf{par}$ is associative and commutative, and work modulo a structural congruence as usual; the details are omitted since they are 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

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

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 Pitts and Starks's ν-calculus [32] 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 $a\#Y \vdash (\nu[a]X)\{b\mapsto Y\} \to \nu[a](X\{b\mapsto Y\})$.

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 scoped 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 3.9, 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.
- By induction on s using the highly syntax-directed nature of the deduction rules, and using the first part.

$\square$

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 set of rewrite rules.

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, $\Delta, \langle a@s\rangle_{nf} \vdash a@t$. We use similar reasoning to deduce $\Delta, \langle a\#s\rangle_{nf} \vdash a@t$, 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. *If $\Delta \vdash s \to t$ then there is a 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 $a \in s$, provided that Δ has enough fresh atoms with respect to the atoms and variables in s, t.*

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

For example, using the notation of this lemma: consider a uniform rewrite system with the trivial rewrite rule $\emptyset \vdash X \to X$ (which is uniform). If $s \equiv \mathsf{И}a.X$ and $\Delta \equiv b\#X$ (which has enough fresh atoms for a and X, namely b) then $t = \mathsf{И}a.X$ and $t' = \mathsf{И}b.(b\ a)\cdot X$.

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

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 a valid pair $\Delta \vdash s \to t_1, t_2$ 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. $l_1|_{p,A}\ {}_?{\approx}_?\ l_2$ has solution (Γ, θ) so $\Gamma \vdash l_1|_{p,A}\theta \approx_\alpha l_2\theta$.
4. $\langle \nabla_i\theta\rangle_{nf}$ is consistent for $i = 1, 2$; write ∇'_i respectively.

Then call the pair of terms-in-context

$$\nabla'_1, \nabla'_2, \Gamma \vdash (r_1\theta, l_1[r_2\theta]_{p,A})$$

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 in l_1 then we say the critical pair is **trivial**.

The condition $\langle \nabla_i\theta\rangle_{nf}$ be consistent is important. $b\#X \vdash f(X) \to X$ 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 $\langle b\#f(b)\rangle_{nf} = \{b\#b\}$ is not consistent.

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 u exists such that $\Delta \vdash t \to^* u$ and $\Delta \vdash t' \to^* u$. 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 u exists such that $\Delta \vdash t \to^* u$ and $\Delta \vdash t' \to^* u$.

Trivial critical pairs are not necessarily joinable in NRSs. For instance with 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 the condition $b@X$ replacing $b\#X$ in this example.

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

LEMMA 6.4. *In a uniform rewrite system trivial critical pairs are joinable.*

PROOF. Suppose there is a trivial critical pair

$$\nabla'_1, \nabla'_2, \Gamma \vdash (r_1\theta, l_1[r_2\theta]_{p,A})$$

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 scope 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. $\square$

This is the reason we are interested in critical pairs:

THEOREM 6.5 (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:

1. There exist $R_i = \nabla_i \vdash l_i \to r_i$ and positions p_i, A_i in s, for $i = 1, 2$.
2. There exist solutions σ_i to $(\nabla \vdash (s[l_i]_{p_i,A_i}, s[r_i]_{p_i,A_i}))\ {}_?{\approx}\ (\Delta \vdash (s, t_i))$ 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 [2]. We need Corollary 3.4 to account for the weaker notion of 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 3.4.

   There are two possibilities: Either this is a non-trivial critical pair, joinable by assumption, or it is a trivial critical pair, we use Lemma 6.4

$\square$

In fact, we need to study rule F a little 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 6.6. *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 $b\#s \vdash (a\ b)\cdot(b\ c)\cdot s \approx_\alpha (a\ c)\cdot s$.* □

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 $\mathcal{R}$ but 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 [29] 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 6.7. *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 rewrite systems in the previous sense [14], 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 [13, 23] (see also [24, 27]).

Say a rule $R \equiv \Delta \vdash l \to r$ is **left-linear** when each variable occuring in l occurs only once. For example, $a\#X, b\#X \vdash X \to (X, X)$ 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 6.8. *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:

$$\frac{\Delta \vdash s \xrightarrow{F}{}^* t}{\Delta \vdash s \Rightarrow t}\,(\text{refl}) \qquad \frac{\Delta \vdash (s_i \Rightarrow t_i)_{1\leq i\leq n}}{\Delta \vdash (s_1,\dots,s_n) \Rightarrow (t_1,\dots,t_n)}\,(\text{tup})$$

$$\frac{\Delta \vdash (s_i \Rightarrow t_i)_{1\leq i\leq n} \quad \Delta \vdash (t_1,\dots,t_n) \xrightarrow{R_\epsilon}\xrightarrow{F}{}^* u}{\Delta \vdash (s_1,\dots,s_n) \Rightarrow u}\,(\text{tup'})$$

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

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

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

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

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.

In Lemma 6.11 we say an instance of (tup'), (abs'), (new'), or (fun') is **for** R when it uses that rule.

LEMMA 6.9. $\xrightarrow{F}{}^* \Rightarrow \xrightarrow{F}{}^* \subseteq \Rightarrow$.

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 6.6. □

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

LEMMA 6.10. *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 $\Delta \vdash s \xrightarrow{\mathcal{R}}{}^* t$.*

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

LEMMA 6.11. *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 u such that $\Delta \vdash t \Rightarrow u$ and $\Delta \vdash t' \Rightarrow u$. 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 u_i such that $\Delta \vdash t_i \Rightarrow u_i$ and $\Delta \vdash t'_i \Rightarrow u_i$. We use (tup) and are done.

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

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

We now proceed as illustrated and explained below:

$$\begin{tikzcd}
(s_1,\dots,s_n) \arrow[r, Rightarrow] \arrow[d, Rightarrow] & (t'_1,\dots,t'_n) \arrow[d, Rightarrow] & \approx_\alpha & l\theta \arrow[r, "R_\epsilon"] & r\theta \arrow[r, "F"] & {}^* t' \arrow[d, Rightarrow] \\
(t_1,\dots,t_n) \arrow[r, Rightarrow] & (u_1,\dots,u_n) & \approx_\alpha & l\theta' \arrow[rr, "R_\epsilon"] & & r\theta'
\end{tikzcd}$$

We apply the inductive hypothesis to close $\Delta \vdash t_i, t'_i \Rightarrow u_i$, using Lemma 6.9 ($\xrightarrow{F}{}^* \Rightarrow \xrightarrow{F}{}^* \subseteq \Rightarrow$) to ensure the tags in u_i are sufficiently fresh that Lemma 6.10 ($\to^* = \Rightarrow$) and Lemma 6.3 can be used to deduce of u_i all freshness and locality assumptions deducible of t'_i.

Since rules are left-linear R still applies: $\Delta \vdash (u_1, \dots, u_n) \xrightarrow{R_\epsilon} r\theta'$ and $\Delta \vdash (t_1, \dots, t_n) \Rightarrow r\theta'$ by (tup) for R (for some substitution θ'). Finally, we use orthogonality, Lemma 6.9, and Lemma 6.6 (F is reversible) to close with a rewrite $t' \Rightarrow r\theta'$.

(3) If $\Delta \vdash s \Rightarrow t'$ is derived using (refl) then trivially $s \equiv t'$ and $\Delta \vdash t' \Rightarrow t$ using the same rules as were used to derive $\Delta \vdash s \Rightarrow t$, and $\Delta \vdash t \Rightarrow t$ is derivable using (refl).

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

We now come back to our theorem:

PROOF. If the uniform rewrite system has only left-linear rules and no non-trivial critical pairs, then $\Rightarrow$ is confluent by Lemma 6.11. Since $\to^* \subseteq \Rightarrow$ and $\to \subseteq \to^*$ by Lemma 6.10, $\to$ 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 rewriting. As an example, we will show how to define in this framework the system λ_{ca} of closed reduction for the λ-calculus [16]. λ_{ca}-terms are λ-terms with explicit constructs for substitutions, copying and erasing. Reduction on λ_{ca} is defined in [16] 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. In effect we need $A(t,\Delta)$ *and* one fresh atom; if $\Delta \vdash a\#t$ for $a \notin A(t,\Delta)$ 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

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

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):

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

8. CONCLUSIONS AND FUTURE WORK

The technical foundations of this work are derived from work on nominal logic [31] and nominal unification [36]. Nominal rewriting was presented in [14]. Here we extended nominal terms and the nominal unification algorithm with a new construct to model scope 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 in a natural way 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 И; there is no space here (we treated orthogonality instead) but the issue needs to be addressed for implementability. 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 (ИA), 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. И.

In conclusion we would like to ask: What are the properties of name-abstraction versus those of name-generation? The theory of $\approx_\alpha$ for name-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 name-generation is generated by $a@$И$a.X$ and $a\#X \vdash X \approx_\alpha$ И$a.X$. (This statement could easily be made formal.)

It is certainly not the case that $a\#X \vdash X \approx_\alpha [a]X$. It is also not the case that $a@X, b@X \vdash X \approx_\alpha (a\ b)\cdot X$. (Though rule F gives us something similar, by allowing us to rename tagged atoms.) So abstraction and name-generation are different. How do they interact in our system?

И 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 И (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 authors' principal technical difficulties designing the system.

Can И model abstraction, or vice versa? Abstraction cannot directly model И, because И does not abstract and of course abstraction does. The converse does not hold, since

$\mathsf{И}$ with α-renaming rules could model the abstraction part of abstraction, but we still have $a@X \vdash X \approx_\alpha \mathsf{И}a.X$, 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 $\mathsf{И}$ directly and saved ourselves the rewrite $\nu[a]X \to \mathsf{И}a.X$:

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

We *cannot* model the λ-calculus 'abstract a in t' as $\lambda\mathsf{И}a.t$, since $\lambda\mathsf{И}a.b \approx_\alpha \lambda b$ is provable. We can model $\nu a.\, t$ in the ν-calculus with $\mathsf{И}$ as $\nu\mathsf{И}a.t$, the reaction is $(\lambda[a]X)Y \to X\{a{\mapsto}Y\}$.

9. REFERENCES

[1] M. Abadi, L. Cardelli, P-L. Curien, and J-J. Lévy, *Explicit substitutions*, Journal of Functional Programming **1** (1991), no. 4, 375–416.

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

[3] F. Barbanera, M. Fernández, and H. Geuvers, *Modularity of strong normalization in the algebraic-λ-cube*, Journal of Functional Programming **6** (1997), 613–660.

[4] H. P. Barendregt, *Pairing without conventional constraints*, Zeitschrift für mathematischen Logik und Grundlagen der Mathematik **20** (1974), 289–306.

[5] C. Bertolissi, H. Cirstea, and C. Kirchner, *Translating combinatory reduction systems into the rewriting calculus*, RULE 2003, Valencia, Spain, 2003.

[6] E. Bonelli, D. Kesner, and A. Ríos, *From higher-order to first-order rewriting*, LNCS vol. 2051, 2001.

[7] V. Breazu-Tannen and J. Gallier, *Polymorphic rewriting conserves algebraic strong normalization*, Theoretical Computer Science 83:1 (1991).

[8] V. Breazu-Tannen and J. Gallier, *Polymorphic rewriting conserves algebraic confluence*, Information and Computation 82 (1992), 3–28.

[9] J. Cheney, *The complexity of equivariant unification.* Proceedings of ICALP 2004, LNCS, 2004.

[10] 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.

[11] 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.

[12] R. David and B. Guillaume, *A λ-calculus with explicit weakening and explicit substitution*, Mathematical Structure in Computer Science **11(1)** (2001), 169–206.

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

[14] 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.

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

[16] M. Fernández, I. Mackie and F-R. Sinot, *Closed Reduction: Explicit Substitutions without alpha-conversion*, MSCS 2005 (to appear).

[17] M. J. Gabbay, *The π-calculus in FM*, Thirty-five years of Automath (Fairouz Kamareddine, ed.), Kluwer, 2003.

[18] 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.

[19] M. Hamana, *Term rewriting with variable binding: An initial algebra approach*, PPDP 2003.

[20] J.-P. Jouannaud and C. Kirchner, *Solving equations in abstract algebras: a rule based survey of unification*, Computational Logic: Essays in Honor of Alan Robinson, 1991.

[21] J.-P. Jouannaud and M. Okada, *Executable higher-order algebraic specification languages*, 6th IEEE Symposium on Logic in Comp. Sci., 1991, pp. 350–361.

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

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

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

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

[26] P. Lescanne, *From λσ to λυ a journey through calculi of explicit substitutions*, POPL'94, 1994.

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

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

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

[30] B. Pagano, *Des calculs de substitution explicite et de leur application à la compilation des langages fonctionnels*, Ph.D. thesis, Université de Paris 6, 1998.

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

[32] A. M. Pitts and I. D. B. Stark, *Observable Properties of Higher Order Functions That Dynamically Create Local Names, or: What's new?.* LNCS, vol. 711, 1993, pp 122–141.

[33] A. M. Pitts and M. J. Gabbay, *A metalanguage for programming with bound names modulo renaming*, MPC2000, LNCS 1837, 2000, pp. 230–255.

[34] F. van Raamsdonk, *Confluence and normalisation for higher-order rewriting*, Ph.D. thesis, Free University of Amsterdam, 1996.

[35] M. R. Shinwell, A. M. Pitts, and Murdoch Gabbay, *FreshML: Programming with binders made simple*, ICFP 2003, pp. 263–274.

[36] C. Urban, A. M. Pitts, and M. J. Gabbay, *Nominal unification*, CSL'03 & KGC. LNCS 2803, 2003, pp. 513–527.

[37] C. Urban, *Nominal matching*, in preparation, 2004.