Two-level lambda-calculus

Murdoch J. Gabbay

http://www.gabbay.org.uk

Dominic P. Mulligan

http://www.macs.hw.ac.uk/~dpm8

Abstract

Two-level lambda-calculus is designed to provide a mathematical model of capturing substitution, also called instantiation. Instantiation is a feature of the 'informal meta-level'; it appears pervasively in specifications of the syntax and semantics of formal languages.
The two-level lambda-calculus has two levels of variable. Lambda-abstraction and beta-reduction exist for both levels. A level 2 beta-reduct, triggering a substitution of a term for a level 2 variable, does not avoid capture for level 1 abstractions. This models meta-variables and instantiation as appears at the informal meta-level.
In this paper we lay down the syntax of the two-level lambda-calculus; we develop theories of freshness, alpha-equivalence, and beta-reduction; and we prove confluence.
In doing this we give nominal terms unknowns — which are level 2 variables and appear in several previous papers — a functional meaning. In doing this we take a step towards longer-term goals of developing a foundation for theorem-provers which directly support reasoning in the style of nominal rewriting and nominal algebra, and towards a mathematics of functions which can bind names in their arguments.

Keywords: Lambda Calculus, Meta-variables, Functional programming, Confluence, Nominal terms

1 Introduction

The λ-calculus is a syntax to express function abstraction and application which is simple, amenable to mathematical analysis, and easy to extend. Yet not everything that looks like a function fits obviously into the λ-calculus; examples include meta-variables, capturing substitution, and functions depending on intensional properties like free variables and freshness conditions associated with free variables.

This paper defines a λ-calculus which contains a model of λ-abstraction and β-reduction for meta-variables, including capturing substitution and freshness conditions.

The idea of internalising meta-variables and capturing substitution into formal languages is a recurrent theme in the literature, though our application of nominal techniques to the problem is new and specific to our work. Models of meta-variables and capturing substitution appear in Jojgov's thesis and subsequent work [20,22], in Munõz's thesis [25], in work by Hashimoto and Ohori [21], by Lee and Friedman [5], and by Kahrs [23]. This is also a theme in the first author's research [7,9,11].

Authors cite various motivations for internalising meta-variables and studying capturing substitution; each author comes to the problem with their own individual background, though the 'convergent evolution' — the product is always

some form of capturing substitution! — suggests that an underlying mathematical entity is being uncovered. These motivations include representing intermediate proof-states (i.e. states of a theorem-prover as it builds a derivation as in the case of Muñoz [25]), and representing 'λ-terms with holes' as a foundation for describing distributed programs, modules, linking, and other manipulations of 'program fragments' ([21,5] belong to this camp and both cogently describe their vision). A comparison with related work is in the Conclusions.

We also think some straightforward mathematical curiosity is called for: capture-avoiding substitution is well-studied and finds expression in many formal logics and calculi, yet capturing substitution is fundamental — see the discussion of the informal meta-level in the next paragraph, or the discussions in the papers by the authors cited above — and it has received relatively less formal attention.

We come to this material from the first author's investigation of nominal rewriting with Fernández [8] and of nominal algebra with Mathijssen [14,24] as mathematical foundations. These use *nominal terms* [29,8,14] (see also later in this paper) advantageously, as a syntax for rewrite rules and algebraic axioms, which very closely model rules and axioms in what we shall call the *informal meta-level* — the prose discourse of, for example, this paper. In the informal meta-level, capturing substitution and freshness conditions are widespread. Here are some examples:

- λ-calculus: $(\lambda x.r)[y \mapsto t] = \lambda x.(r[y \mapsto t])$ if x is fresh for t
- π-calculus: $\nu x.(P \mid Q) = P \mid \nu x.Q$ if x is fresh for P
- First-order logic: $\forall x.(\phi \supset \psi) = \phi \supset \forall x.\psi$ if x is fresh for ϕ

These quoted (reified) informal statements mention two levels of variable; *object-variables* x, y and *meta-variables* r, t, P, Q, ϕ, ψ. Capture-avoidance conditions are (freshness) constraints on the values that meta-variables may assume.

It seems that meta-variables are naturally substituted with capturing substitution; consider the following quote:

"Set r to x and t to x in $(\lambda x.r)[y \mapsto t]$; obtain $(\lambda x.x)[y \mapsto x]$."

Capturing substitution on syntax trees is not hard to define, yet syntax usually has semantics, which motivates the study of (amongst other things) α-equivalence, unification, and β-equivalence.

Nominal terms feature a two-level hierarchy of variables directly reflecting the hierarchy noted above:

- **level 1** variables $a, b, c, d, \ldots$ (atoms) model object-variables;
- **level 2** variables $X, Y, Z, \ldots$ (unknowns) model meta-variables.

Variables in nominal terms have no in-built functional meaning. There is no λ-abstraction λX or λa, and no application. In a suitable (generalised) sense, nominal terms are first-order. Indeed, until now the semantics for level 2 variables (unknowns) in nominal terms has been 'they range over terms'. This purely syntactic meaning is acceptable for unification, matching, and rewriting [29,8], and

possibly for logic-programming [3], but taking a broader view this is clearly only a partial answer. In this paper we extend nominal terms to investigate full $\alpha\beta$-equivalence in the presence of level 1 and level 2 variables (thus: $\alpha\beta$-equivalence in the presence of object- and meta-variables). By giving level 2 variables a notion of λ-abstraction and β-reduction, we give them a functional operational semantics.

We present **two-level λ-calculus**. We give a syntax, reduction system, and prove confluence and consistency. The calculus contains the untyped λ-calculus (two copies, in fact; at level 1 and level 2), so it is not strongly normalising.

Just as the step from first-order terms to higher-order terms implies an increase in expressivity traded against the loss of some computational and mathematical properties [31], so we expect similar benefits and drawbacks moving from nominal terms to two-level λ-calculus — but the λ-calculus now expresses both capturing and capture-avoiding substitution.

We sketch with examples the kind of thing we can do, or believe should be possible in future work, using two-level λ-calculus. Syntax and full definitions follow.

"Set t to be x in $\lambda x.t$" and "set t to be y in $\lambda x.t$" are modelled by reductions

$$(\lambda X.(\lambda a.X))a \to (\lambda a.X)[X := a] \equiv \lambda a.a \quad \text{and}$$

$$(\lambda X.(\lambda a.X))b \to (\lambda a.X)[X := b] \equiv \lambda a.b.$$

($\equiv$ is syntactic equivalence, i.e. 'the same term'.) Capture-avoidance for λa (level 1) in the presence of level 2 variables is managed using permutations and freshness in nominal terms style:

$$c\#X \vdash (\lambda b.(\lambda a.X))a \to \lambda c.(((c\ a) \cdot X)[b \mapsto a]).$$

"Set r to x and t to x in $(\lambda x.r)[y{\mapsto}t]$; obtain $(\lambda x.x)[y{\mapsto}x]$" is expressed by:

"$\lambda Z.\lambda Y.((\lambda a.Z)[b \mapsto Y])$ applied to a twice reduces to $(\lambda a.a)[b \mapsto a]$."

It is not hard to impose a simple type system on our syntax (this should be a special case of [6]; here, λX should cause no essential new difficulties). An interesting application for our calculus is as the basis of a logic. Example axioms are (we indicate types with subscripts):

- $\forall P_o.(a_o\#P_o \supset P_o \supset \forall a_o.P_o)$
  Here o is a type of truth-values. $\forall$ is short for $\forall\lambda$ where $\forall$ is a constant symbol. $\#$ is short for $\#\lambda$ where $\#$ is a constant symbol intended to internalise the nominal freshness judgement. This models 'for all ϕ, if $a \notin fv(\phi)$ then $\phi \supset \forall a.\phi$'.
- $\forall X_\alpha.(a_\beta\#X_\alpha \supset \lambda a_\beta.(X_\alpha a_\beta) = X_\alpha)$
  Here $=$ is a constant symbol, written infix. α and β are intended to be arbitrary types. This models η-equivalence (extensionality) at level 1.
- $\forall P_o.(И a_{\mathbb{A}}.\neg P_o) \Leftrightarrow \neg И a_{\mathbb{A}}.P_o$.
  Here И is short for Иλ where И is a constant symbol intended to internalise the Gabbay-Pitts 'new' quantifier [19]. $\neg$ and $\Leftrightarrow$ are constant symbols. $\mathbb{A}$ is a 'type of atoms' with no term-formers. This models the self-duality of И.

The reductions are real reductions in our calculus. The axioms are expressed in a logic which does not yet exist, but they still have some mathematical force because the structure they express has been studied in previous work with level 2 variables but (since nominal terms have no λX) without a level 2 quantification explicitly represented in the syntax. The literature contains theories (sets of axioms) for several specific object languages, including first-order logic, the π-calculus, and subsets of ML [24,19,8,11]. It also contains a study describing how to use a fragment of the calculus in this paper, as the term-language for a Curry-Howard correspondence for 'incomplete derivations' [13] (level 2 variables represent 'unknown proofs', as in Muñoz [25]). Authors mentioned above have their own reasons for considering multi-level systems [20,22,25,21,5,23] and we currently see no obvious reason why the ideas in this paper might not also be turned to other applications.

2 The syntax

Definition 1 Fix disjoint countably infinite sets of **level 1** and **level 2** variables. We let $a, b, c, \ldots$ and $X, Y, Z, \ldots$ range over level 1 and level 2 variables respectively. We use a **permutative convention** that these are distinct. For example, 'a and b' means any pair of distinct level 1 variables. [1]

Remark 2 Level 2 variables are sometimes called *meta-variables* (as in for example [25]). We prefer not to do this; level 2 variables are designed to model meta-variables, but they inhabit the syntax and are objects in the formal syntax. The 'real' meta-level still exists — for example we are writing in it now — and mixing terminology for the real thing and our formal model of it, seems to us too great a possible source of confusion.

Definition 3 A **permutation** π is a *finitely supported* bijection of level 1 variables. Here 'finitely supported' means that $\pi(a) = a$ for all but finitely many level 1 variables.

Write id for the identity permutation, so $id(a) = a$ always; write $\circ$ for functional composition, so

$$(\pi \circ \pi')(a) = \pi(\pi'(a)).$$

Write π^{-1} for inverse. Write $(a\ b)$ for the **swapping** such that

$$(a\ b)(a) = b, \quad (a\ b)(b) = a, \quad \text{and} \quad (a\ b)(c) = c.$$

Definition 4 Let the syntax of two-level λ-calculus be:

$$r, s, t, u, v \ ::= \ a \mid \pi \cdot X \mid \lambda a.r \mid \lambda X.r \mid rr$$

Intuitively, this is two copies of the λ-calculus; a level 1 copy with a, rr, and $\lambda a.r$, and a level 2 copy with $\pi \cdot X$, rr, and $\lambda X.r$. Both copies have capture-avoiding substitution within their own level. The challenge is to understand how these two copies interact; we discuss this in the rest of this section.

[1] a and b are meta-variables ranging permutatively over level 1 variables. X and Y are meta-variables ranging permutatively over level 2 variables. However, we will probably just call a a level 1 variable, and X a level 2 variable, just as we may later call r a term when in fact it is a meta-variable ranging over terms (Definition 4).

$$a[X := t] \equiv a \qquad (\pi \cdot X)[X := t] \equiv \pi \cdot t \qquad (\pi \cdot Y)[X := t] \equiv \pi \cdot Y$$

$$(\lambda a.r)[X := t] \equiv \lambda a.(r[X := t]) \qquad (r'r)[X := t] \equiv (r'[X := t])(r[X := t])$$

$$(\lambda Y.r)[X := t] \equiv \lambda Y.(r[X := t]) \ \ (Y \notin fv(t))$$

$$\pi \cdot a \equiv \pi(a) \qquad \pi \cdot (\pi' \cdot X) \equiv (\pi \circ \pi') \cdot X \qquad \pi \cdot (r'r) \equiv (\pi \cdot r')(\pi \cdot r)$$

$$\pi \cdot (\lambda a.r) \equiv \lambda \pi(a).(\pi \cdot r) \qquad \pi \cdot (\lambda X.r) \equiv (\lambda X.\pi \cdot r[X := \pi^{-1} \cdot X])$$

Figure 1. Level 2 substitution and level 1 permutation action (Definition 6)

We equate terms up to α-equivalence of λX-bound variables, but *not* λa-bound variables (this is not necessary, but it is convenient). We write $\equiv$ for syntactic equivalence (up to α-equivalence of λX-bound variables). For example,

$$\lambda X.X \equiv \lambda Y.Y \quad \text{but} \quad \lambda a.X \not\equiv \lambda b.X \quad \text{and} \quad \lambda a.a \not\equiv \lambda b.b.$$

There is no paradox here. We do not seek to eliminate the meta-level; we want mathematical models with which to study it. That is why we have given our calculus two levels of variable.

We may write

$$r[a \mapsto t] \quad \text{as shorthand for} \quad (\lambda a.r)t.$$

Definition 5 Define a notion $fv(r)$ of **free occurrence** (for level 2) by:

$$fv(a) = \emptyset \qquad fv(\pi \cdot X) = \{X\} \qquad fv(r'r) = fv(r') \cup fv(r)$$

$$fv(\lambda a.r) = fv(r) \qquad fv(\lambda X.r) = fv(r) \setminus \{X\}$$

Definition 6 Define **level 2 substitution** $r[X := t]$ and **level 1 permutation** actions by the rules in Figure 1.

Level 2 substitution avoids capture by λ-abstracted level 2 variables, but not by λ-abstracted level 1 variables. Compare the clauses for $(\lambda a.r)[X := t]$ and $(\lambda Y.r)[X := t]$:

- $(\lambda a.r)[X := t] \equiv \lambda a.(r[X := t])$ always (no condition that 'a should be fresh for t').
- $(\lambda Y.r)[X := t] \equiv \lambda Y.(r[X := t])$ provided $Y \notin fv(t)$ (a freshness condition).

From the point of view of level 1 variables, level 2 variables behave like meta-variables and are instantiated. The usual apparatus of level 2 β-conversion gives us the power to program 'normally' at level 2.

The level 1 permutation action $\pi \cdot r$ is the general mechanism by which level 1 α-renaming is managed; it is based on ideas inherited from nominal terms [29]. It is not always the case that $\pi \cdot r$ is α-equivalent with r, but in specific special cases where all variables that π permutes are bound, this is the case; more on this in Section 3. For example,

$$(a\ b) \cdot \lambda a.\lambda b.ab \equiv \lambda b.\lambda a.ba.$$

For a discussion of the advantages of this approach, see Cheney's comment that permutations are 'inherently capture-avoiding' [2] and the surrounding discussion.

$\pi \cdot X$ is a *moderated* level 2 variable; this is the point in the raw syntax where the two levels of variable interact. Moderations rename level 1 variables in uninstantiated level 2 variables; for example:

$$((b\ a) \cdot X)[X{:=}a] \equiv b$$

In Definition 6 we must decide how the permutation action $\pi \cdot r$ interacts with level 2 λ-abstraction λX (this is a new issue which does not appear in nominal terms, because they do not have level 2 λ-abstraction). The intuition is 'π moderates free but not bound level 2 variables'. For example:

$$\pi \cdot (XY) \equiv (\pi \cdot X)(\pi \cdot Y)$$

$$\pi \cdot \lambda X.(XY) \equiv \lambda X.X\pi \cdot Y$$

$$\pi \cdot \lambda Y.\lambda X.(XY) \equiv \lambda Y.\lambda X.(XY).$$

$Y \notin fv(t)$ in the clause for $(\lambda Y.r)[X := t]$ can be guaranteed by renaming Y.

The definitions of $r[X := t]$ and $\pi \cdot r$ are intertwined. This is due to the clause for $\pi \cdot (\lambda X.r)$. The proof that $\pi \cdot r$ and $r[X := t]$ are well-defined is routine, and we omit it. It uses a notion of **depth** of a term, which will be useful later:

$$depth(a) = 1 \qquad depth(\pi \cdot X) = 1 \qquad depth(\lambda a.r) = 1 + depth(r)$$

$$depth(\lambda X.r) = 1 + depth(r) \qquad depth(r'r) = depth(r') + depth(r)$$

Lemma 7 $r[X := \pi \cdot X][X := \pi' \cdot X] \equiv r[X := (\pi \circ \pi') \cdot X]$

Lemma 8 $\pi \cdot (\pi' \cdot r) \equiv (\pi \circ \pi') \cdot r$

Lemma 9 $\pi \cdot (r\sigma) \equiv (\pi \cdot r)\sigma$

Proof We prove the result for $\sigma = [Y := u]$ by induction on r. We consider just the case of $\lambda X.r$: α-converting X if necessary, assume $X \notin fv(u)$. Then:

$$(\pi \cdot \lambda X.r)[Y := u] \equiv \lambda X.(\pi \cdot r[X := \pi^{-1} \cdot X][Y := u])$$

$$\pi \cdot ((\lambda X.r)[Y := u]) \equiv \pi \cdot (\lambda X.r[Y := u]) \equiv \lambda X.(\pi \cdot r[X := \pi^{-1} \cdot X][Y := u]).$$

□

Lemma 10 $depth(r) = depth(\pi \cdot r)$.

3 Freshness and reductions

3.1 Definition of the rules

Definition 11 We define some notation:

$$\frac{}{\Delta \vdash a\#b}\ (\mathbf{a\#b}) \qquad \frac{}{\Delta \vdash a\#\lambda a.r}\ (\mathbf{a\#\lambda a}) \qquad \frac{\Delta \vdash a\#r}{\Delta \vdash a\#\lambda b.r}\ (\mathbf{a\#\lambda b})$$

$$\frac{\pi^{-1}(a)\#X \in \Delta}{\Delta \vdash a\#\pi\cdot X}\ (\mathbf{a\#X})$$

$$\frac{\Delta \vdash a\#r' \quad \Delta \vdash a\#r}{\Delta \vdash a\#r'r}\ (\mathbf{a\#app}) \qquad \frac{\Delta, a\#X \vdash \pi(a)\#\pi\cdot r \quad (X \notin \Delta)}{\Delta \vdash \pi(a)\#\pi\cdot(\lambda X.r)}\ (\mathbf{a\#\lambda X})$$

Figure 2. Freshness entailment (Definition 11)

- A **freshness** is a pair $a\#r$ of a level 1 variable and a term.
- Call $a\#X$ a **primitive freshness**.
- Call a finite set of primitive freshnesses Δ a **freshness context**.

We may drop set brackets, for example writing $\{a\#X, b\#X\}$ as $a\#X, b\#X$ and writing $\Delta, a\#X$ for $\Delta \cup \{a\#X\}$. We may also write $a\#X, b\#X$ as $a, b\#X$.

Figure 2 gives freshnesses a notion of derivation.

The notion of nominal freshness is well-discussed [19,29,24]. Intuitively,

$$a\#r \quad \text{corresponds with } `a \notin fv(r)',$$

read 'a is abstracted/not free in r'. In the presence of level 2 variables, which can be instantiated (substituted in a possibly capturing manner) to any term, the notion of 'free variables' is replaced by a notion of freshness which may depend on freshness assumptions on level 2 variables. For example:

$$\frac{\dfrac{\dfrac{}{a\#X, a\#Y \vdash a\#X}\ (\mathbf{a\#X}) \quad \dfrac{}{a\#X, a\#Y \vdash a\#Y}\ (\mathbf{a\#X})}{a\#X, a\#Y \vdash a\#XY}\ (\mathbf{a\#app})}{a\#Y \vdash b\#\lambda X.(X(b\ a)\cdot Y)}\ (\mathbf{a\#\lambda X}) \qquad \frac{\dfrac{}{a\#X \vdash a\#\lambda a.X}\ (\mathbf{a\#\lambda a})}{\vdash a\#\lambda X.\lambda a.X}\ (\mathbf{a\#\lambda X})$$

We may write '$\Delta \vdash a\#t$' as shorthand for '$\Delta \vdash a\#t$ is derivable'. We use this shorthand for other derivable assertions including $\rightarrow$, $\rightarrow^*$, $\Rightarrow$, $\Rightarrow^*$, defined later.

Remark 12 $(\mathbf{a\#\lambda X})$ does not implement the denotational notion of freshness for functions from nominal sets [19], with intuition "$(b\ a)\cdot\lambda X.r$ equals $\lambda X.r$ for fresh b". Informally, a is fresh for the function '$\lambda X.\lambda a.X$' in the sense of $(\mathbf{a\#\lambda X})$, but not fresh in the sense of nominal sets. $(\mathbf{a\#\lambda X})$ is a strictly weaker condition; a discussion of denotations is for a later paper.

Remark 13 $(\mathbf{a\#\lambda X})$ is syntax directed, in the following sense:

$$depth(\pi\cdot\lambda X.r) = depth(r) + 1 > depth(r) = depth(\pi\cdot r)$$

See Lemma 10.

Definition 14 Write $\Delta \vdash \text{-} \triangleright \text{-}$ for a binary relation on terms parameterised on Δ. Call $\Delta \vdash \text{-} \triangleright \text{-}$ a **congruence** when it is closed under the rules named $\triangleright$ in Figure 3.

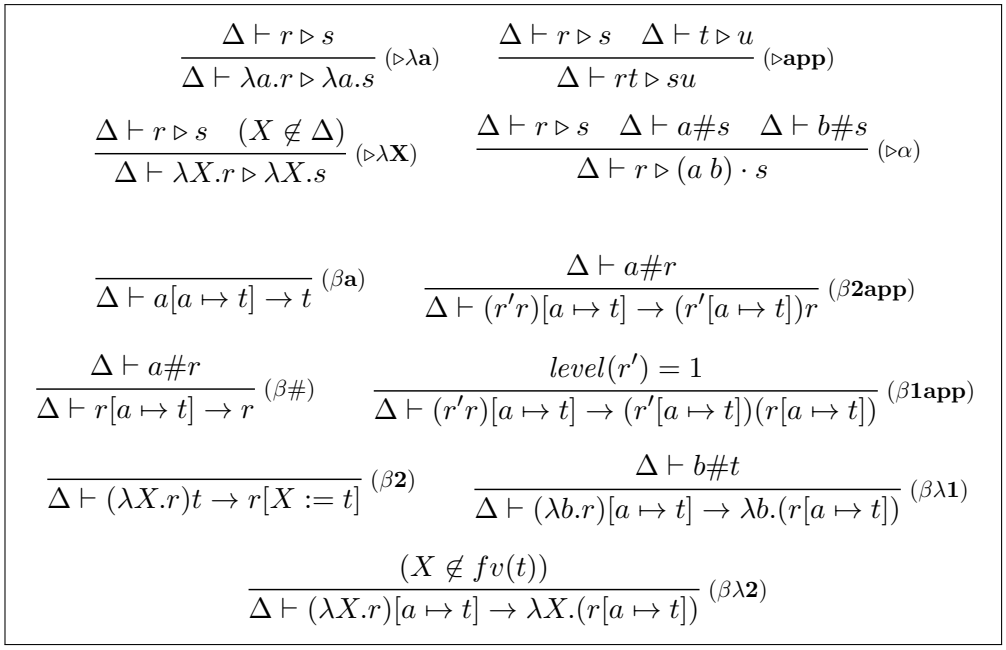


Figure 3. Congruence (Definition 14) and reductions (Definition 17)

Remark 15 $(\triangleright\alpha)$ builds in the nominal term notion of α-equivalence for level 1 variables (see [14, Lemma 3.2]). A better name for congruence might be 'congruent α-equivalence', but we just write 'congruence'.

Definition 16 Write $level(t) = 2$ if t mentions a level 2 variable, free or bound. Otherwise, write $level(t) = 1$. For example

$$level((a\ b)\cdot X) = level(\lambda X.X) = 2 \quad \text{and} \quad level(a) = level(\lambda a.a) = 1.$$

Write $a \notin \Delta$ when $a\#X \notin \Delta$ for all level 2 variables X. Write $X \notin \Delta$ when $a\#X \notin \Delta$ for all level 1 variables a.

We can now define our reduction relation:

Definition 17 Let $\Delta \vdash - \to -$ be the least congruence closed under the rules named β in Figure 3.

Recall that $r[a \mapsto t]$ is sugar for $(\lambda a.r)t$.

Remark 18 Level 2 β-reduction is a single step $[X := t]$, level 1 β-reduction is not. Definition 27 gives the single step level 1 β-reduction. There are no level 3 variables in this particular calculus and therefore no special need arises to break level 2 β-reduction into smaller steps.

Example reductions are in the Introduction. Note that capture-avoiding substitution is as usual within a single level:

$$(\lambda b.(\lambda a.b))a \to^* \lambda a'.(b[b \mapsto a]) \to \lambda a'.a$$

$$(\lambda Y.(\lambda X.Y))X \to \lambda X'.(Y[Y := X]) \equiv \lambda X'.X$$

3.2 *Discussion of rules* $(\beta 1app)$ *and* $(\beta 2app)$

$(\beta\mathbf{1app})$ and $(\beta\mathbf{2app})$ can be viewed as two parts of a single rule $(\beta\mathbf{app})$:

$$\frac{\Delta \vdash a\#r \text{ or } level(r') = 1}{\Delta \vdash (r'r)[a \mapsto t] \to (r'[a \mapsto t])(r[a \mapsto t])}\ (\beta\mathbf{app})$$

If $level(r') = 1$ *and* $\Delta \vdash a\#r$ we may join $(\beta\mathbf{1app})$ and $(\beta\mathbf{2app})$ with $(\beta\#)$, like so:

$$\begin{array}{ccc} (r'r)[a \mapsto t] & \xrightarrow{(\beta\mathbf{2app})} & (r'[a \mapsto t])r \\ \Big\downarrow{\scriptstyle (\beta\mathbf{1app})} & & \Big\updownarrow{\scriptstyle *} \\ (r'[a \mapsto t])(r[a \mapsto t]) & \xrightarrow{(\beta\#)} & (r'[a \mapsto t])r \end{array}$$

The restrictions in $(\beta\mathbf{1app})$ and $(\beta\mathbf{2app})$ are necessary. Suppose we drop them, to obtain just one rule

$$(r'r)[b \mapsto u] \overset{(\mathbf{FALSE})}{\longrightarrow} (r'[b \mapsto u])(r[b \mapsto u]).$$

Then for example: [2]

$$\begin{array}{rl} ((\lambda X.X[b \mapsto a])b)[b \mapsto c] & \overset{(\mathbf{FALSE})}{\longrightarrow} (\lambda X.X[b \mapsto a])[b \mapsto c](b[b \mapsto c]) \\ & \overset{(\beta\lambda\mathbf{2}),(\beta\mathbf{a})}{\longrightarrow} (\lambda X.X[b \mapsto a][b \mapsto c])c \\ & \overset{(\beta\mathbf{2})}{\longrightarrow} c[b \mapsto a][b \mapsto c] \\ & \overset{(\beta\#),(\beta\#)}{\longrightarrow} c \end{array}$$

$$\begin{array}{rl} ((\lambda X.X[b \mapsto a])b)[b \mapsto c] & \overset{(\beta\mathbf{2})}{\longrightarrow} b[b \mapsto a][b \mapsto c] \\ & \overset{(\beta\mathbf{1})}{\longrightarrow} a[b \mapsto c] \\ & \overset{(\beta\#)}{\longrightarrow} a \end{array}$$

Thus, confluence fails in the presence of $(\mathbf{FALSE})$.

Intuitively, level 2 and level 1 β-reducts can 'compete' to instantiate variables. The side-conditions on $(\beta\mathbf{1app})$ and $(\beta\mathbf{2app})$ prevent this competition from destroying confluence. Intuitively level 2 β-reductions 'happen first' — though note that, subject to the side-conditions, β-reducts can reduce in any order in the two level λ-calculus.

More formally, to close a divergence in $((\lambda X.r)t)[b \mapsto u]$ between $(\beta\mathbf{2})$ and $(\mathbf{FALSE})$ we must join

$$r[X := t][b \mapsto u] \quad \text{and} \quad r[b \mapsto u][X := t[b \mapsto u]],$$

[2] Thanks to Stéphane Lengrand for helping develop this pithy example.

where $X \notin fv(u)$ and $X \notin fv(t)$ and where the freshness context Δ is such that $\Delta \vdash b\#u$. It is not possible to join this in general, as the example above shows. It is possible to join this if $level(r) = 1$ or if $\Delta \vdash b\#t$.

The interested reader might like to compare the side-conditions in $(\beta\mathbf{1app})$ and $(\beta\mathbf{2app})$ with the side-condition on the corresponding rule $(\sigma\mathbf{p})$ in the lambda-context calculus [11, Figure 5]. For the reader's convenience we reproduce it here (a_i denotes a variable of level i; the lambda context calculus has an infinite hierarchy of levels):

$$(ss')[a_i \mapsto t] \to (s[a_i \mapsto t])(s'[a_i \mapsto t]) \qquad (level(s), level(s'), level(t) \leq i)$$

Transposing this to the two-level lambda-calculus we would obtain a rule

$$\frac{level(r') = level(r) = level(t) = 1}{\Delta \vdash (r'r)[a \mapsto t] \to (r'[a \mapsto t])(r[a \mapsto t])}\ (\beta\mathbf{LCC})$$

Clearly, $(\beta\mathbf{1app})$ and $(\beta\mathbf{2app})$ allow more reductions than $(\beta\mathbf{LCC})$. The technical reason that we are able to allow these reductions turns out to be the stronger theory of α-equivalence in this paper. Specifically, we should look at the renaming to fresh c in the case of $(\lambda b.r)[a := t]$ if $\Delta \nvdash b\#t$ in Definition 27. That renaming is not in general possible in the lambda-context calculus; the lambda-context calculus lacks freshness contexts and permutations, so it is not always possible to generate a fresh c and rename a level 1 binding to our fresh c using a permutation. More comment on the lambda-context caluclus is in the Conclusions.

3.3 *Substitution and soundness*

In the rest of this section we prove soundness results; that freshness and reduction are preserved under instantiating level 2 variables (Lemma 20 and Theorem 24), and that freshness is preserved under reduction (Theorem 22):

Definition 19 A **substitution** σ is a finitely supported map from level 2 variables to terms. 'Finitely supported' means that $\sigma(X) \equiv id \cdot X$ for all but finitely many variables. These act on terms $t\sigma$ in the natural way, extending the action of $[X := t]$ from Definition 6. We extend this action pointwise as convenient; in particular we write $\Delta\sigma$ for $\{a\#\sigma(X) \mid a\#X \in \Delta\}$.

If $\mathcal{F}$ is a set of freshnesses write $\Delta' \vdash \mathcal{F}$ for '$\Delta' \vdash a\#r$ for every $a\#r \in \mathcal{F}$'. Lemma 20 is soundness for substitutions compatible with the freshness context:

Lemma 20 *If $\Delta' \vdash \Delta\sigma$ then $\Delta \vdash a\#r$ implies $\Delta' \vdash a\#(r\sigma)$.*

Proof By induction on r. We consider two cases:

– The case of λa. $\Delta' \vdash a\#(\lambda a.r)\sigma$ always, by $(\mathbf{a\#\lambda a})$.

– The case of λX. Suppose $\Delta, a\#X \vdash \pi(a)\#\pi \cdot r$ is derivable, where $X \notin \Delta$. Suppose the inductive hypothesis of the derivation. Renaming X if necessary, we may assume that $X \notin \Delta'$ and $\sigma(X) \equiv X$. By inductive hypothesis $\Delta', a\#X \vdash \pi(a)\#(\pi \cdot r)\sigma$ is derivable. By Lemma 9, $(\pi \cdot r)\sigma \equiv \pi \cdot (r\sigma)$. We extend the derivation with $(\mathbf{a\#\lambda X})$ to obtain a derivation of $\Delta' \vdash \pi(a)\#\pi \cdot (\lambda X.(r\sigma))$. We

then have $\pi \cdot (\lambda X.(r\sigma)) \equiv \pi \cdot ((\lambda X.r)\sigma) \equiv (\pi \cdot \lambda X.r)\sigma$ (the final $\equiv$ uses Lemma 9), and the result follows.

□

Lemma 21 *$\Delta \vdash a\#r$ implies $\Delta \vdash \pi(a)\#\pi \cdot r$.*

Proof By induction on derivations, we consider just one case:

– The case $(\mathbf{a\#\lambda X})$. Suppose $\Delta, a\#X \vdash \pi(a)\#\pi \cdot r$. By inductive hypothesis $\Delta, a\#X \vdash \pi'(\pi(a))\#\pi' \cdot (\pi \cdot r)$, so $\Delta, a\#X \vdash (\pi' \circ \pi)(a)\#(\pi' \circ \pi) \cdot r$ by Lemma 8. Using $(\mathbf{a\#\lambda X})$ we derive $\Delta \vdash (\pi' \circ \pi)(a)\#(\pi' \circ \pi) \cdot \lambda X.r$. We use Lemma 8 to deduce $\Delta \vdash \pi'(\pi(a))\#\pi' \cdot (\pi \cdot \lambda X.r)$.

□

Theorem 22 is 'subject-reduction for freshness':

Theorem 22 *If $\Delta \vdash r \to s$ then $\Delta \vdash a\#r$ implies $\Delta \vdash a\#s$.*

Proof By induction on derivations. We consider a selection of cases:

– The case $(\beta\mathbf{2app})$, assuming $\Delta \vdash b\#r$. Suppose $\Delta \vdash b\#r$ and $\Delta \vdash (r'r)[b \mapsto u] \to (r'[b \mapsto u])r$.

If $\Delta \vdash a\#(r'r)[b \mapsto u]$ then $\Delta \vdash a\#r'$, $\Delta \vdash a\#r$, and $\Delta \vdash a\#u$. It follows that $\Delta \vdash a\#(r'[b \mapsto u])r$. Similarly if $\Delta \vdash b\#(r'r)[b \mapsto u]$. Similarly for $(\beta\mathbf{1app})$.

– The case $(\beta\mathbf{2})$. Suppose that $\Delta \vdash (\lambda X.r)t \to r[X := t]$.

If $\Delta \vdash a\#(\lambda X.r)t$ then $\Delta \vdash a\#t$ and $\Delta, a\#X \vdash a\#r$. It follows by Lemma 20 that $\Delta \vdash a\#r[X := t]$ as required.

– The case $(\triangleright\alpha)$. Suppose that $\Delta \vdash r \to s$ and $\Delta \vdash a\#s$ and $\Delta \vdash b\#s$.

$\Delta \vdash a\#s$ and $\Delta \vdash b\#s$ by assumption. If $\Delta \vdash c\#r$ then, by inductive hypothesis, $\Delta \vdash c\#s$, and so $\Delta \vdash c\#(a\ b) \cdot s$ by Lemma 21.

□

Lemma 23 *If $level(r) = 1$ then $level(r\sigma) = 1$.*

Theorem 24 does for $\to$ what Lemma 20 did for freshness; it is a form of soundness under substitutions compatible with the freshness contexts:

Theorem 24 *If $\Delta' \vdash \Delta\sigma$ then $\Delta \vdash r \to s$ implies $\Delta' \vdash r\sigma \to s\sigma$.*

Proof By induction on derivations. We present a selection of cases.

– The case $(\beta\#)$. Suppose $\Delta \vdash a\#r$. By Theorem 22 $\Delta' \vdash a\#r\sigma$ if $\Delta \vdash a\#r$. It follows by $(\beta\#)$ that $\Delta' \vdash r\sigma[a \mapsto t\sigma] \to r\sigma$.

– The case $(\beta\mathbf{1app})$. We have $(r'r)[a \mapsto t]\sigma \equiv (r'\sigma)(r\sigma)[a \mapsto t\sigma]$. By assumption $level(r') = 1$ so by Lemma 23, $level(r'\sigma) = 1$ and $\Delta' \vdash (r'\sigma)(r\sigma)[a \mapsto t\sigma] \to (r'[a \mapsto t])(r[a \mapsto t])\sigma$ by $(\beta\mathbf{1app})$. The result follows.

– The case $(\beta\lambda\mathbf{1})$. By Theorem 22 if $\Delta \vdash b\#t$ then $\Delta' \vdash b\#t\sigma$. It follows by $(\beta\lambda\mathbf{1})$ that $\Delta' \vdash (\lambda b.r\sigma)[a \mapsto t\sigma] \to \lambda b.(r\sigma[a \mapsto t\sigma])$. As $\lambda b.(r\sigma[a \mapsto t\sigma]) \equiv (\lambda b.r[a \mapsto t])\sigma$, the result follows.

□

4 Confluence

Our calculus can be viewed as two λ-calculi (level 1, level 2) glued together by a nominal treatment of the interaction of level 1 α-equivalence with level 2 variables. The proof of confluence also splits into two proofs, plus some 'proof-glue'.

4.1 Level 1 reductions

Definition 25 Let (**level1**) be the set

$$\{(\beta\mathbf{a}), (\beta\#), (\beta\mathbf{1app}), (\beta\mathbf{2app}), (\beta\lambda\mathbf{1}), (\beta\lambda\mathbf{2})\}$$

of rules from Figure 3. Let $\Delta \vdash r \xrightarrow{(\text{level1})} s$ be the least congruence closed under the rules in (**level1**).

Definition 26 Write $\Delta \nvdash a\#r$ when $\Delta \vdash a\#r$ is *not* derivable.

Choose some fixed but arbitrary order on atoms. If S is a finite set of atoms say 'for the first atom not in S' to mean 'for the least atom, in our fixed but arbitrary order, that is not an element of S'.[3]

Definition 27 For a given Δ, define a **level 1 substitution action** $r[a := t]$ as below; earlier rules take priority:

- $r[a := t] \equiv r$ if $\Delta \vdash a\#r$.
- $a[a := t] \equiv t$.
- $(\pi \cdot X)[a := t] \equiv (\pi \cdot X)[a \mapsto t]$.
- $(r'r)[a := t] \equiv (r'[a := t])(r[a := t])$ provided $level(r') = 1$.
- $(r'r)[a := t] \equiv r'[a := t]r$ provided $\Delta \vdash a\#r$.
- $(r'r)[a := t] \equiv (r'r)[a \mapsto t]$ if $\Delta \nvdash a\#r$ and $level(r') = 2$.
- $(\lambda b.r)[a := t] \equiv \lambda b.(r[a := t])$ provided $\Delta \vdash b\#t$.
- $(\lambda b.r)[a := t] \equiv (\lambda c.((b\ c) \cdot r)[a := t])$ if $\Delta \nvdash b\#t$. Here c is the first atom not mentioned in r, a, b, or t, such that $\Delta \vdash c\#r$ and $\Delta \vdash c\#t$, if such a c exists. Thus we α-convert the level 1 λb to a fresh λc, but in the presence of level 2 variables we use an explicit permutation and explicitly fresh level 1 variable, in nominal terms style.
- $(\lambda b.r)[a := t] \equiv (\lambda b.r)[a \mapsto t]$ otherwise.
- $(\lambda X.r)[a := t] \equiv \lambda X.(r[a := t])$, renaming X if needed so that $X \notin t$.

Then for that Δ, let r^* be defined as follows; earlier rules have priority, read left-to-right then top-to-bottom (Δ will always be clear from the context):

$$a^* \equiv a \quad (\pi \cdot X)^* \equiv \pi \cdot X \quad (\lambda a.r)^* \equiv \lambda a.r^* \quad (\lambda X.r)^* \equiv \lambda X.r^*\ (X \notin \Delta)$$

$$(r'[a \mapsto t])^* \equiv r'^*[a := t^*] \quad ((\lambda X.r)t)^* \equiv r^*[X := t^*] \quad (r'r)^* \equiv r'^*r^*$$

We can always rename X to ensure $X \notin \Delta$.

[3] This is not necessary for expressing the proofs to follow but it is convenient. We will never make infinitely many choices of fresh atom, and nowhere will the truth of a result depend on our choice of order.

– If $\Delta = \{a\#X\}$ then $X[a := b] \equiv X$ and $(X[a := b])^* \equiv X$.
– If $\Delta = \emptyset$ then $X[a := b] \equiv X[a \mapsto b]$.

Intuitively, r^* is a canonical form of r, and $r[a := t]$ is a canonical form of $r[a \mapsto t]$. This is not necessarily a normal form (which may not exist), but garbage is collected and substitutions (level 1 β-reducts) are pushed into r. Level 2 β-reducts are unreduced.

Definition 28 Call $\Delta \vdash \text{-} \triangleright \text{-}$

- **reflexive** when $\Delta \vdash r \triangleright r$ always, and
- **transitive** when $\Delta \vdash r \triangleright r'$ and $\Delta \vdash r' \triangleright r''$ imply $\Delta \vdash r \triangleright r''$.

Write $\Delta \vdash \text{-} \triangleright^* \text{-}$ for the least transitive reflexive relation containing $\Delta \vdash \text{-} \triangleright \text{-}$.

Definition 29 Suppose that Δ and Δ^+ are freshness contexts. Say Δ^+ **freshly extends** Δ when there exists some Δ' such that:

- $\Delta \cap \Delta' = \emptyset$ (in words: Δ and Δ' are disjoint),
- $\Delta^+ = \Delta \cup \Delta'$,
- for all $c\#X \in \Delta'$ it is the case that $c \notin \Delta$, and for all $c\#X \in \Delta'$ it is the case that $X \in \Delta$.

Intuitively, Δ^+ 'extends Δ with some fresh atoms'.

Lemma 30 *For every Δ, r, a, and t there exists a Δ^+ freshly extending Δ such that $\Delta^+ \vdash r[a \mapsto t] \overset{(\textbf{level1})}{\longrightarrow}{}^* r[a := t]$ ($r[a := t]$ calculated for Δ^+.)*

Proof Each rule in Definition 27 is emulated by a rule in (**level1**). We may need some fresh atoms to α-convert. □

Lemma 31 $depth(r) = depth(r[X := \pi \cdot Y])$

Lemma 32 *We have:*

(i) $(\pi \cdot r)^* \equiv \pi \cdot r^*$

(ii) *If $(\pi \cdot t)^* \equiv \pi \cdot t^*$ for all π, then $(r[X := t])^* \equiv r^*[X := t^*]$*

Proof By induction on $depth(r)$, using Lemma 10 and Lemma 31. □

Theorem 33 states that terms reduce to their canonical form; it enters via Lemma 30 that we may need some fresh atoms, to α-convert.

Theorem 33 *For every Δ and r there exists a Δ^+ freshly extending Δ such that $\Delta^+ \vdash r \overset{(\textbf{level1})}{\longrightarrow}{}^* r^*$ (r^* calculated for Δ^+.)*

Proof By induction on r. For example, $a^* \equiv a$ and $(\lambda X.r)^* \equiv \lambda X.r^*$. The special case $(\lambda X.r)t$ requires Lemma 32, and the case $(\lambda a.r)t$ requires Lemma 30. □

Lemma 34 *Fix Δ. Then $\Delta \vdash a\#s$ implies $\Delta \vdash a\#s^*$. (s^* calculated for Δ.)*

Proof By induction on derivations, using Lemma 32. □

Lemma 35 *Suppose $\Delta \vdash r \overset{(\textbf{level1})}{\longrightarrow} s$ and also that $\Delta' \vdash \Delta[X := \pi \cdot X]$, then $\Delta' \vdash r[X := \pi \cdot X] \overset{(\textbf{level1})}{\longrightarrow} s[X := \pi \cdot X]$.*

Proof By induction on derivations, using Lemma 31, Lemma 20 and Lemma 9. □

Lemma 36 *If* $\Delta \vdash r \overset{(\textbf{level1})}{\longrightarrow} s$ *then* $\Delta \vdash \pi \cdot r \overset{(\textbf{level1})}{\longrightarrow} \pi \cdot s$.

Proof By induction on derivations, using Lemma 21 and Lemma 35. □

Theorem 37, along with Theorem 33 above, makes up the diagram in Theorem 38:

Theorem 37 *For every* Δ, r, *and* s, *there exists a* Δ^+ *freshly extending* Δ *such that* $\Delta \vdash r \overset{(\textbf{level1})}{\longrightarrow} s$ *implies* $\Delta^+ \vdash s^* \overset{(\textbf{level1})}{\longrightarrow}{}^* r^*$. *($r^*$ and s^* calculated for Δ^+.)*

Proof By induction on the derivation of $\Delta \vdash r \overset{(\textbf{level1})}{\longrightarrow} s$. We sketch some cases:

– The case $(\beta\textbf{2app})$ where $\Delta \vdash a\#r$. We use Lemma 34 to conclude that $\Delta^+ \vdash a\#r^*$ in the case where $level(r') = 1$.

$$((r'r)[a{\mapsto}t])^* \equiv (r'^*[a{:=}t^*])(r^*[a{:=}t^*]) \equiv (r'^*[a{:=}t^*])(r^*) \qquad \text{if } level(r') = 1$$

$$((r'r)[a{\mapsto}t])^* \equiv (r'^*[a{:=}t^*])(r^*) \qquad \text{if } level(r') = 2$$

– The case $(\beta\textbf{1app})$ where $level(r') = 1$.

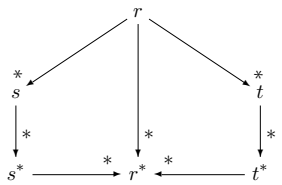
$$((r'[a \mapsto t])(r[a \mapsto t]))^* \equiv (r'^*[a := t^*])(r^*[a := t^*]) \equiv ((r'r)[a \mapsto t])^*$$

– The case $(\beta\lambda\textbf{2})$. Suppose $X \notin t$, which can be guaranteed. Then $\Delta \vdash (\lambda X.r)[a \mapsto t] \overset{(\textbf{level1})}{\longrightarrow} \lambda X.(r[a \mapsto t])$. We have $((\lambda X.r)[a \mapsto t])^* \equiv \lambda X.r^*[a := t^*] \equiv (\lambda X.(r[a \mapsto t]))^*$, and we have the result.
– The case $(\triangleright\alpha)$. Suppose $s \equiv (a\ b) \cdot s'$. Suppose $\Delta \vdash r \to s'$ and $\Delta \vdash a\#s'$ and $\Delta \vdash b\#s'$. By inductive hypothesis, $\Delta^+ \vdash s'^* \overset{(\textbf{level1})}{\longrightarrow}{}^* r^*$. By Lemma 34, $\Delta^+ \vdash a\#s'^*$ and $\Delta^+ \vdash b\#s'^*$. By Theorem 22 it follows that $\Delta^+ \vdash a\#r^*$ and $\Delta^+ \vdash b\#r^*$. By $(\triangleright\alpha)$ it follows that $\Delta^+ \vdash s'^* \overset{(\textbf{level1})}{\longrightarrow}{}^* (a\ b) \cdot (r^*)$. By Lemmas 8 and 36 it follows that $\Delta^+ \vdash (a\ b) \cdot (s'^*) \overset{(\textbf{level1})}{\longrightarrow}{}^* r^*$. By Lemma 32, $\Delta^+ \vdash s^* \overset{(\textbf{level1})}{\longrightarrow}{}^* r^*$ as required.

□

Theorem 38 $(\textbf{level1})$ *is confluent.*

Proof From the following diagram, using Theorem 33 and Theorem 37:



□

4.2 *Level 2*

Definition 39 Let $(\textbf{level2})$ be the set

$$\{(\beta\textbf{2}), (\beta\lambda\textbf{2})\}$$

$$\frac{}{\Delta \vdash a \Rightarrow a}\ (\mathbf{Pa}) \qquad \frac{}{\Delta \vdash (\pi{\cdot}X) \Rightarrow (\pi{\cdot}X)}\ (\mathbf{PX})$$

$$\frac{\Delta \vdash r \Rightarrow s \quad \Delta \vdash t \Rightarrow u}{\Delta \vdash rt \Rightarrow su}\ (\mathbf{Papp}) \qquad \frac{\Delta \vdash r \Rightarrow s \quad \Delta \vdash t \Rightarrow u \quad \Delta \vdash su \xrightarrow{R} v}{\Delta \vdash rt \Rightarrow v}\ (\mathbf{Papp}\varepsilon)$$

$$\frac{\Delta \vdash r \Rightarrow s}{\Delta \vdash \lambda a.r \Rightarrow \lambda a.s}\ (\mathbf{P}\lambda\mathbf{a}) \qquad \frac{\Delta \vdash r \Rightarrow s}{\Delta \vdash \lambda X.r \Rightarrow \lambda X.s}\ (\mathbf{P}\lambda\mathbf{X})$$

$$\frac{\Delta \vdash r \Rightarrow s \quad \Delta \vdash a\#s \quad \Delta \vdash b\#s}{\Delta \vdash r \Rightarrow (a\ b) \cdot s}\ (\mathbf{P}\alpha)$$

Figure 4. The parallel reduction relation (Definition 40)

of rules from Figure 3. Let $\Delta \vdash r \xrightarrow{(\mathbf{level2})} s$ be the least congruence closed under the rules in (**level2**).

Note that $\xrightarrow{(\mathbf{level1})}$ and $\xrightarrow{(\mathbf{level2})}$ have $(\beta\lambda\mathbf{2})$ in common. This is necessary for confluence to work, see Lemma 51.

Definition 40 Define a **parallel reduction relation**, $\Rightarrow$, by the rules in Figure 4. Here, R ranges over rules in (**level2**) (Definition 39).

Lemma 41 *$\Delta \vdash r \Rightarrow s$ implies $\Delta \vdash r \xrightarrow{(\mathbf{level2})}{}^{*} s$.*

As a corollary, $\Delta \vdash r \Rightarrow^{*} s$ implies $\Delta \vdash r \xrightarrow{(\mathbf{level2})}{}^{*} s$.

Lemma 42 *$\Delta \vdash r \xrightarrow{(\mathbf{level2})} s$ implies $\Delta \vdash r \Rightarrow s$.*

As a corollary, $\Delta \vdash r \xrightarrow{(\mathbf{level2})}{}^{*} s$ implies $\Delta \vdash r \Rightarrow^{*} s$

Proof It suffices to show that every possible (**level2**)-reduction can be mirrored by a parallel reduction. This is routine. The second part follows from the first part and an easy proof that $\Delta \vdash r \Rightarrow r$ ($\Rightarrow$ is reflexive). □

Theorem 43 *$\Delta \vdash r \Rightarrow^{*} s$ if and only if $\Delta \vdash r \xrightarrow{(\mathbf{level2})}{}^{*} s$*

Proof From Lemmas 41 and 42. □

Lemma 44 *If $\Delta \vdash r \xrightarrow{(\mathbf{level2})} s$ then $\Delta \vdash \pi \cdot r \xrightarrow{(\mathbf{level2})} \pi \cdot s$.*

Proof By induction on derivations. We mention two cases:

– The case $(\triangleright\alpha)$. By inductive hypothesis, we have $\Delta \vdash \pi \cdot r \to \pi \cdot s$, and from Lemma 21, we have $\Delta \vdash \pi(a)\#\pi \cdot s$ and $\Delta \vdash \pi(b)\#\pi \cdot s$. Using $(\triangleright\alpha)$ we obtain $\Delta \vdash (\pi \cdot r) \to (\pi(a)\ \pi(b)) \cdot (\pi \cdot s)$ and the result follows via elementary properties of permutations.

– The case $(\beta\mathbf{2})$. Using Lemma 7, we have:

$$\pi{\cdot}((\lambda X.r)t) \equiv (\lambda X.\pi{\cdot}r[X{:=}\pi^{-1}{\cdot}X])(\pi{\cdot}t) \to (\pi{\cdot}r[X{:=}\pi^{-1}{\cdot}X])[X{:=}\pi{\cdot}t]$$
$$\equiv \pi{\cdot}(r[X{:=}\pi^{-1}{\cdot}X][X{:=}\pi{\cdot}t]) \equiv (\pi{\cdot}r)[X{:=}t] \equiv \pi{\cdot}(r[X{:=}t])$$

□

Lemma 45 *If* $\Delta \vdash r \Rightarrow s$ *and* $\Delta' \vdash \Delta[X{:=}\pi{\cdot}X]$ *then* $\Delta' \vdash r[X{:=}\pi{\cdot}X] \Rightarrow s[X{:=}\pi{\cdot}X]$.

Proof By induction on derivations using Lemma 9, Lemma 20 and Theorem 24. □

Lemma 46 *If* $\Delta \vdash r \Rightarrow s$ *then* $\Delta \vdash \pi \cdot r \Rightarrow \pi \cdot s$.

Proof The proof is by induction on derivations. We present a selection of cases:

– The case ($\mathbf{P}\lambda\mathbf{X}$). By inductive hypothesis, we have $\Delta \vdash \pi \cdot r \Rightarrow \pi \cdot s$. From this and Lemma 45 we can obtain $\Delta \vdash (\pi \cdot r)[X := \pi^{-1} \cdot X] \Rightarrow (\pi \cdot s)[X := \pi^{-1} \cdot X]$. Using ($\mathbf{P}\lambda\mathbf{X}$), we have $\Delta \vdash \pi \cdot \lambda X.r \Rightarrow \pi \cdot \lambda X.s$.

– The case ($\mathbf{Papp}\varepsilon$). By inductive hypothesis, we have $\Delta \vdash \pi \cdot r \Rightarrow \pi \cdot s$ and $\Delta \vdash \pi \cdot t \Rightarrow \pi \cdot u$, and also $\Delta \vdash (\pi \cdot s)(\pi \cdot u) \overset{(\mathbf{level2})}{\longrightarrow} \pi \cdot v$, using Lemma 44. Then $\Delta \vdash (\pi \cdot r)(\pi \cdot t) \Rightarrow \pi \cdot v$ is derivable, and the result follows.

– The case ($\mathbf{P}\alpha$). By inductive hypothesis, $\Delta \vdash \pi \cdot r \Rightarrow \pi \cdot s$, and by Lemma 21, $\Delta \vdash \pi(a)\#\pi \cdot s$ and $\Delta \vdash \pi(b)\#\pi \cdot s$. Then $\Delta \vdash \pi \cdot r \Rightarrow (\pi(a)\ \pi(b)) \cdot (\pi \cdot s)$ is derivable and the result follows by elementary properties of permutations.

□

Lemma 47 $\Delta \vdash r \Rightarrow s$ *and* $\Delta \vdash t \Rightarrow u$ *imply* $\Delta \vdash r[X := t] \Rightarrow s[X := u]$.

Proof The proof is by induction on the derivation of $\Delta \vdash r \Rightarrow s$ (Figure 4). We assume that $\Delta \vdash t \Rightarrow u$.

– The case ($\mathbf{PX}$). By Lemma 46, and by the fact that $(\pi \cdot Y)[X := t] \equiv \pi \cdot Y$.

– The case ($\mathbf{Papp}\varepsilon$). Suppose $\Delta \vdash r \Rightarrow s$, $\Delta \vdash v \Rightarrow w$ and $\Delta \vdash sw \overset{(\mathbf{level2})}{\longrightarrow} v$. Then $\Delta \vdash r[X := t] \Rightarrow s[X := u]$ and $\Delta \vdash v[X := t] \Rightarrow w[X := u]$ are both derivable, and from Theorem 24, we have $\Delta \vdash s[X := u](w[X := u]) \rightarrow v[X := u]$. The result follows from ($\mathbf{Papp}\varepsilon$).

– The case ($\mathbf{P}\alpha$). By inductive hypothesis, $\Delta \vdash r[X := t] \Rightarrow s[X := u]$, and by Lemma 20, $\Delta \vdash a\#s[X := u]$ and $\Delta \vdash b\#s[X := u]$. Extending with ($\mathbf{P}\alpha$), we obtain $\Delta \vdash r[X := t] \Rightarrow (a\ b) \cdot (s[X := t])$. The result follows from Lemma 9.

□

Write

$$\text{`}\Delta \vdash r \Leftarrow s \Rightarrow t\text{'} \text{ for } \text{`}\Delta \vdash s \Rightarrow r \text{ and } \Delta \vdash s \Rightarrow t\text{'},$$

and similarly for other reduction relations later.

Lemma 48 $\Delta \vdash \text{-} \Rightarrow \text{-}$ *satisfies the diamond property: If* $\Delta \vdash r \Leftarrow s \Rightarrow t$ *then there exists some* u *such that* $\Delta \vdash r \Rightarrow u \Leftarrow t$.

Proof We consider possible pairs of rules that can derive $\Delta \vdash s \Rightarrow r$ and $\Delta \vdash s \Rightarrow t$, where $r \not\equiv t$. We present the case of ($\mathbf{Papp}$) and ($\mathbf{Papp}\varepsilon$) for ($\beta\mathbf{2}$). Suppose $\Delta \vdash (\lambda X.r')t' \Leftarrow (\lambda X.r)t \Rightarrow r''[X := t'']$ using ($\mathbf{Papp}$), and ($\mathbf{Papp}\varepsilon$) for ($\beta\mathbf{2}$). By inductive hypothesis, there exists an r''' such that $\Delta \vdash r' \Rightarrow r''' \Leftarrow r''$ and a t''' such that $\Delta \vdash t' \Rightarrow t''' \Leftarrow t''$. Using Lemma 47 we derive $\Delta \vdash (\lambda X.r')t' \Rightarrow r'''[X := t'''] \Leftarrow r''[X := t'']$. □

Theorem 49 $\overset{(\mathbf{level2})}{\longrightarrow}$ *is confluent.*

Proof An immediate corollary of Theorem 43 and Lemma 48. □

4.3 Level 1 and level 2 reductions

Lemma 50 $\Delta \vdash r \to s$ *implies* $level(s) \leq level(r)$, *and so does* $\Delta \vdash r \Rightarrow s$.

Lemma 51 $\Delta \vdash r \Leftarrow s \overset{(\mathbf{level1})}{\longrightarrow} t$ *implies that* $\Delta \vdash r \overset{(\mathbf{level1})}{\longrightarrow}{}^{*} u \Leftarrow t$. *As a corollary,* $\Delta \vdash r \Leftarrow s \overset{(\mathbf{level1})}{\longrightarrow}{}^{*} t$ *implies* $\Delta \vdash r \overset{(\mathbf{level1})}{\longrightarrow}{}^{*} u \Leftarrow t$.

Proof We assume $\Delta \vdash u \Rightarrow u'$, $\Delta \vdash t \Rightarrow t'$, and so on. We present some cases:

$-\ \Delta \vdash (r'''r'')[a \mapsto t'] \Leftarrow (r'r)[a \mapsto t] \to (r'[a \mapsto t])(r[a \mapsto t])$ when $level(r') = 1$.
This can be closed to $(r'''[a \mapsto t'])(r''[a \mapsto t'])$; by Lemma 50 $level(r''') = 1$.
$-\ \Delta \vdash (r'''r'')[a \mapsto t'] \Leftarrow (r'r)[a \mapsto t] \to (r'[a \mapsto t])r$ when $\Delta \vdash a\#r$.
This can be closed to $(r'''[a \mapsto t'])r''$.
$-\ \Delta \vdash ((\lambda X.r')t')[b \mapsto u'] \Leftarrow ((\lambda X.r)t)[b \mapsto u] \to ((\lambda X.r)[b \mapsto u])t$ when $\Delta \vdash b\#t$.
This can be closed to $((\lambda X.r')[b \mapsto u'])t'$.
$-\ \Delta \vdash r'[X := t'][b \mapsto u'] \Leftarrow ((\lambda X.r)t)[b \mapsto u] \to ((\lambda X.r)[b \mapsto u])t$ when $\Delta \vdash b\#t$.
This can be closed to $r'[X := t'][b \mapsto u']$ where $X \notin u$ which can be guaranteed; we require $(\beta\lambda\mathbf{2})$ in $(\mathbf{level2})$.

□

Lemma 52 *If* $\Delta \vdash r \overset{(\mathbf{level1})}{\longleftarrow} s \to^{*} t$, *then* $\Delta^{+} \vdash r \to^{*} u \longleftarrow t$ *for some suitably freshened context,* Δ^{+}. *Similarly, if* $\Delta \vdash r \overset{(\mathbf{level2})}{\longleftarrow} s \to^{*} t$, *then* $\Delta^{+} \vdash r \to^{*} u \longleftarrow t$ *for some suitable freshened context* Δ^{+}.

Proof Both claims follow by induction on the path length of $\Delta \vdash s \to^{*} t$ using Theorem 38, Theorem 49 and Lemma 51. □

Theorem 53 $\Delta \vdash \text{-} \to \text{-}$ *(reduction with* $(\mathbf{level2}) \cup (\mathbf{level1})$*) is confluent, in the following sense: if* $\Delta \vdash s \to^{*} r$ *and* $\Delta \vdash s \to^{*} t$ *then there exists some* Δ^{+} *freshly extending* Δ, *and some* u, *such that* $\Delta^{+} \vdash r \to^{*} u$ *and* $\Delta^{+} \vdash t \to^{*} u$.

Proof By a diagrammatic argument, using Lemma 52. □

The 'freshly extending' part of Theorem 53 ensures we can α-rename λa with a permutation to guarantee the freshness precondition of $(\beta\lambda\mathbf{1})$. In the presence of only a single level of variable we can 'just rename', and so we do not need to record the freshness of freshly generated variables relative to higher-level variables in a freshness context.

Recall Definition 28. Call $\Delta \vdash \text{-} \triangleright \text{-}$ **symmetric** when $\Delta \vdash r \triangleright r'$ implies $\Delta \vdash r' \triangleright r$. Write $\Delta \vdash \text{-} = \text{-}$ for the least congruence closed under $\Delta \vdash \text{-} \to \text{-}$ that is also transitive, reflexive, and symmetric.

Corollary 54 $\Delta \vdash \text{-} = \text{-}$ *is consistent. (There exist two terms which are not related by the transitive reflexive symmetric closure of* $\Delta \vdash \text{-} \to \text{-}$.*)*

Proof $\lambda a.\lambda b.a$ and $\lambda a.\lambda b.b$ are two distinct terms, and they do not rewrite to a common term. By Theorem 53 the result follows. □

5 Related and future work

Previous 'nominal' λ-calculi.

The NEW calculus of contexts and the lambda-context calculus [9,11] are λ-calculi with more than one level of variable. These have a weak theory of α-equivalence which is less powerful than that of nominal terms. There are no permutations and it is not possible to α-rename a in $\lambda a.X$, where a has level 1 and X has level 2. This problem carries through to the theory of reduction since β-reductions can get stuck due to lack of α-equivalence. For example, $(\lambda a.X)[b \mapsto a]a$ cannot reduce because we cannot α-convert λa to λc for a fresh c. (A similar observation goes back at least to [5].)

The NEW calculus of contexts and the lambda-context calculus retain some unique features not subsumed by this paper. They have an infinite hierarchy of levels and, arguably, are closer to the λ-calculus (precisely because their syntax does not include permutations). It is future work to combine β-conversion for more than two levels of variable with the notion of reduction with nominal terms style α-equivalence appearing in this paper.

The lambda-context calculus preserves strong normalisation. A variant of two-level λ-calculus which preserves strong normalisation is possible using ideas from [11]. The proof of confluence is not harder. The result in this paper is more relevant for designing logics, where we trade off more reductions against weaker computational properties.

Capture-avoiding substitution as a nominal algebra with Mathijssen [12,18] used nominal terms syntax, with permutations and freshness; substitution corresponds to level 1 β-reductions. There is no level 2 λ-abstraction λX.

λ-calculi for capturing substitution.

The λ-calculus itself can emulate capturing substitution. Let r and t range over λ-terms: then the quote in the Introduction is emulated by

'apply $\lambda x.((\lambda y.r)(yx))$ to $\lambda x.t$'.

However, compositionality fails in the sense that the number of 'extra' λ-abstractions needed on t, and 'extra' applications needed on y, depend non-locally on the variables for which we authorise capture at the point(s) at which y occurs in r. See [26, Section 2] for further discussion.

λ-calculi exist with more than one level of variable, or with constructions with essentially the same intent. We mentioned several in the Introduction.

Jojgov and Geuvers. Jojgov and Geuvers [20] study incomplete derivations in higher-order logic. They create o-HOL, which conservatively extends HOL with 'open' terms and 'open' (incomplete) proofs. 'Meta-variables' (level 2 variables in our terminology) are represented in o-HOL syntax. Their treatment of α-conversion is not nominal, but neither is it 'traditional'. The definition of meta-variable instantiation on page 546 (page 10 of the paper, just after Definition 11) in simplified form reads:

$$\underline{n}[q]\{\underline{n}[y] := t\} = t[q/y]$$

Here $\underline{n}$ is a meta-variable symbol (level 2 variable), q and t are terms, and y is a(n ordinary, level 1) variable. Meta-variable symbols are always associated either with a (list of) variable(s) y or a (list of) term(s) q. By this rather ingenious device Jojgov and Geuvers seem to manage α-equivalence in the presence of two levels of variable. $\underline{n}[q]$ seems to correspond with what we might write '$X[\text{-} \mapsto q]$' and the meta-variable instantiation $\{\underline{n}[y] := t\}$ seems to correspond with what we might write '$[y \mapsto \text{-}][X := t]$'.

Establishing the precise technical relationship between our term language and that of Jojgov and Geuvers might be the topic of future work (though the approach of Jojgov and Geuvers has not been under development for some time). However, our syntax seems to be simpler and more atomic. Level 2 variables exist 'as is' (not annotated by suspended substitutions). In our system level 1 substitution is managed very simply by level 1 β-reducts, whereas Jojgov and Geuvers have two copies of level 1 substitution; level 1 β-reducts *and* the variable-and-term annotations on the meta-variables.

Pfenning et al. A thread of research exemplified by [26] manages capturing vs. non-capturing substitution inside a specially-designed logical framework. Our calculus uses levels; there is no (need for a) 'wrapping' in formal logic and the meaning of a term is contained in the term itself, rather than in the term and its interaction with the 'wrapping' logical framework. Also, our level 2 variables are 'open' (instantiation captures unless a freshness condition forbids it) whereas context variables in [26] are 'closed' (variables that can be captured must be accounted for, in the type system).

Sato et al. Sato *et al* have investigated calculi with meta-variables. Their calculi are very similar to the lambda-context calculus but forbid β-reduction if terms contain variables that are 'too strong'. In particular $(\lambda a.(Z'Z))Y$ will not β-reduce in their system [28, Subsection 3.3]. For better or for worse, this is a clearly weaker notion of reduction (there are fewer reductions) than two-level lambda-calculus (it is also weaker than the notion of reduction of the lambda-context calculus).

We also mention work by Hashimoto and Ohori [21], which annotates level 2 variables with 'variable renamers' similar to our permutations, and work by Lee and Friedman [5] which is more like a full meta-programming system but with an emphasis on meta-variables.

Our two-level λ-calculus is specifically tailored to nominal terms and so fits into our broader research programme. The syntax is relatively simple, and we may well be able to to import other nominal research, from semantics [19,10] to implementation [30]. Thus, we have not yet built semantics or a type system for two-level λ-calculus, but tools exist to do this in a principled way [10,6].

Future work.

The complexity of presentation of the two-level lambda-calculus compares reasonably favourably with that of the systems we find by other authors in the existing literature, for example [25,21,20]. We may be able to simplify the presentation futher, this is future work.

We intend to impose types and create a higher-order logic, as outlined in the Introduction. A discussion will follow once the logic is created. Nominal unification,

nominal algebra, and one-and-a-halfth order logic [29,24,17] have already done much of the work but the universal quantification of level 2 variables (unknowns) X is implicit (there is no λX or $\forall X$). Denotations, presumably using nominal sets [19], are future work. We can also consider enriching our functional operational semantics with nominal unification, which is computationally more tractable than higher-order unification [29].

A word on terminology

We have created several multi-level systems: nominal terms [29], hierarchical nominal rewriting [7], the NEW calculus of contexts [9] and the lambda-context calculus [11], one-and-a-halfth-order logic [15] and nominal algebra [14,24]. They all have a capturing substitution but differ in complexity, scale, and detail. We need a taxonomy to describe them.

We have used '-and-a-halfth order', as in 'one-and-a-halfth order logic' [17] or 'two-and-a-halfth order lambda-calculus' [16]. We have found that this sometimes causes confusion because 'order' is usually associated with types. Levels can be viewed as a sort system, but they are not types in the sense of simple type theory [4] (for example, in the two-level lambda-calculus X and a have different levels but can be substituted for exactly the same terms, namely, any terms at all).

In this paper we propose and use the following terminology:

- One level system. Examples include the typed and untyped lambda-calculus, and first- and higher-order logic.
- One-and-a-half level system. Two levels of variable, abstraction only for level 1 (not for level 2; i.e. a, λa, and X, but no λX). Examples include nominal terms, nominal unification, nominal rewriting, nominal algebra, and one-and-a-halfth order logic (which we would now call one-and-a-half level logic).
- Two level system. Two levels of variable, abstraction for both. The calculus in this paper is a two level lambda-calculus with a nominal terms style theory of level 1 alpha-equivalence (until now, we would call it 'two-and-a-halfth order lambda-calculus'; now we call it the 'two-level lambda-calculus').
- Multi-level system. An infinity of levels with abstraction for all levels. Examples include hierarchical nominal rewriting, the NEW calculus of contexts, and the lambda-context calculus. See [11,9] for accounts of how having more than two levels of variable can be useful.

References

[1] Mirna Bognar. *Contexts in Lambda Calculus*. PhD thesis, Vrije Universiteit Amsterdam, 2002.

[2] James Cheney. Nominal logic and abstract syntax. *SIGACT News (logic column 14)*, 36(4):47–69, 2005.

[3] James Cheney and Christian Urban. αProlog: A logic programming language with names, binding and alpha-equivalence. In Bart Demoen and Vladimir Lifschitz, editors, *Proc. of the 20th Int'l Conf. on Logic Programming (ICLP 2004)*, number 3132 in LNCS, pages 269–283. Springer-Verlag, 2004.

[4] Alonzo Church. A formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5(2):56–68, 1942.

[5] Shinn-Der Lee and Daniel P. Friedman. Enriching the lambda calculus with contexts: toward a theory of incremental program construction. In ICFP '96: Proceedings of the first ACM SIGPLAN international conference on functional programming, pages 239–250, 1996.

[6] Maribel Fernández and Murdoch J. Gabbay. Curry-style types for nominal terms. In *Types for Proofs and Programs*, volume 4502/2007 of *LNCS*, pages 125–139, 2006.

[7] Murdoch J. Gabbay. Hierarchical nominal rewriting. In *LFMTP'06: Logical Frameworks and Meta-Languages: Theory and Practice*, pages 32–47, 2006.

[8] Maribel Fernández and Murdoch J. Gabbay. Nominal rewriting. *Information and Computation*, 205(6):917–965, 2007.

[9] Murdoch J. Gabbay. A NEW calculus of contexts. In *PPDP'05*, pages 94–105. ACM, 2005.

[10] Murdoch J. Gabbay. A General Mathematics of Names. *Information and Computation*, 205:982–1011, July 2007.

[11] Murdoch J. Gabbay and Stéphane Lengrand. The lambda-context calculus. *ENTCS*, 196:19–35, 2008.

[12] Murdoch J. Gabbay and Aad Mathijssen. Capture-avoiding Substitution as a Nominal Algebra. In *ICTAC*, volume 4281 of *LNCS*, pages 198–212, 2006.

[13] Murdoch J. Gabbay and Dominic P. Mulligan. One-and-a-halfth order terms: Curry-Howard for incomplete derivations. In *WoLLIC '08: International Workshop on Logic, Language, Information and Computation*, volume 5110 of *LNAI*, pages 180–194, 2008.

[14] Murdoch J. Gabbay and Aad Mathijssen. A formal calculus for informal equality with binding. In *WoLLIC'07: 14th Workshop on Logic, Language, Information and Computation*, volume 4576 of *LNCS*, pages 162–176, 2007.

[15] Murdoch J. Gabbay and Aad Mathijssen. One-and-a-halfth order Logic. In *PPDP '06: Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 189–200, 2006.

[16] Murdoch J. Gabbay and Dominic P. Mulligan. Two-and-a-halfth order lambda-calulus. In *WFLP '08: 17th International Workshop on Functional and (Constraint) Logic Programming*, preliminary proceedings, pages 107–121, 2008.

[17] Murdoch J. Gabbay and Aad Mathijssen. One-and-a-halfth-order Logic (journal version). *Journal of Logic and Computation*, November 2007. Online.

[18] Murdoch J. Gabbay and Aad Mathijssen. Capture-avoiding Substitution as a Nominal Algebra (journal version). *Formal Aspects of Computing*, 2008. Online.

[19] Murdoch J. Gabbay and A. M. Pitts. A New Approach to Abstract Syntax with Variable Binding (journal version). *Formal Aspects of Computing*, 13(3–5):341–363, 2001.

[20] Herman Geuvers and Gueorgui I. Jojgov. Open proofs and open terms. In *CSL '02: Proceedings of the 16th International Workshop and 11th Annual Conference of the EACSL on Computer Science Logic*, pages 537–552, 2002.

[21] Masatomo Hashimoto and Atsushi Ohori. A typed context calculus. *Theoretical Computer Science*, 266(1-2):249–272, 2001.

[22] Gueorgui I. Jojgov. Holes with binding power. In *Types for Proofs and Programs*, volume 2646 of *LNCS*, pages 162–181. Springer, 2002.

[23] Stefan Kahrs. Context rewriting. In *CTRS '92: 3rd International Workshop on Conditional Term Rewriting Systems*, volume 656 of *LNCS*, pages 21–35, 1992.

[24] Aad Mathijssen. *Logical Calculi for Reasoning with Binding*. PhD thesis, Technische Universiteit Eindhoven, 2007.

[25] César Muñoz. A Calculus of Explicit Substitutions for Incomplete Proof Representation in Type Theory. PhD thesis, INRIA Rocqencourt, Projet Coq, 1997.

[26] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *Transactions on Computational Logic*, 2007.

[27] Masahiko Sato, Takafumi Sakurai, and Rod Burstall. Explicit environments. *Fundamenta Informaticae*, 45:1-2:79–115, 2001.

[28] Masahiko Sato, Takafumi Sakurai, Yukiyoshi Kameyama, and Atsushi Igarashi. Calculi of meta-variables. In *CSL*, volume 2803 of *LNCS*, pages 484–497, 2003.

[29] C. Urban, A. M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.

[30] Christian Urban and Christine Tasson. Nominal techniques in Isabelle/HOL. In *CADE '05: Proceedings of the 20th International Conference on Automated Deduction*, volume 3632 of *Lecture Notes in Artificial Intelligence*, pages 38–53, 2005.

[31] Johan van Benthem. Higher-order logic. In *Handbook of Philosophical Logic, 2nd Edition*, volume 1, pages 189–244. Kluwer, 2001.