Permissive-Nominal Logic

Gilles Dowek
www.lix.polytechnique.fr/~dowek

Murdoch J. Gabbay
www.gabbay.org.uk

Abstract

Permissive-Nominal Logic (PNL) is an extension of first-order logic where term-formers can bind names in their arguments.

This allows for direct axiomatisations with binders, such as the $\forall$-quantifier of first-order logic itself and the λ-binder of the lambda-calculus. This also allows us to finitely axiomatise arithmetic.

Like first- and higher-order logic and unlike other nominal logics, equality reasoning is not necessary to α-rename.

All this gives PNL much of the expressive power of higher-order logic, but terms, derivations and models of PNL are first-order in character, and the logic seems to strike a good balance between expressivity and simplicity.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic*]: Proof theory—Nominal techniques

General Terms Theory

Keywords First-order logic, permissive-nominal terms, mechanized mathematics.

1. Introduction

Binding is ubiquitous in logical specifications in mathematics — it features in function definitions as λ-abstraction, and binders are also used to define sets in comprehension, and to define finite and infinite sums, integrals, derivations, and so on.

This suggests we consider logics that support term-formers that can bind variables, to conveniently specify systems with binding. This turns out to be not so easy.

First-order logic is computationally tractable (unification of first-order terms is decidable, and proof-search is simple and well-understood) and expressive (it can be used for example to specify groups, rings, and combinators).

However, first-order logic does not admit term-formers that can bind. Thus it is difficult in first-order logic to give direct, finite, axiomatisations of set theory, arithmetic, higher-order logic, or the λ-calculus.

This is one reason that e.g. higher-order logic is often used as a specification language in theory (see [9] for an excellent exposition) and implementations (like Isabelle [27]). Higher-order logic has a convenient and powerful binder (λ) built in to terms. However, it is also stronger than first-order logic, less computationally tractable, and models tend to be more complex. The jump from first- to higher-order is quite large.

This motivates the study of direct extensions of first-order logic with term-formers that can bind. The topic of this paper is the construction of one such extension, which we call permissive-nominal logic (PNL). PNL has a clear first-order flavour, and it admits term-formers that bind.

An overview of the content of this paper is as follows:

- We introduce the syntax and derivations of permissive-nominal logic (Section 2).
- We axiomatise equality, substitution, first-order logic, 'nominal' inductive datatypes, the И-quantifier, semantic freshness — and arithmetic (Sections 3 and 7).
- We prove soundness (Theorems 5.15) for a suitable semantics of permissive-nominal sets (Section 4).
- We prove the axiomatisation of arithmetic correct (Theorem 6.20).

It is a fact that Cut can be eliminated and PNL is complete for the semantics in this paper. Proofs will be in a longer paper.

Unification of permissive-nominal terms is decidable [7], and models of PNL are like those of first-order logic. Complexities common in higher-order syntax and semantics do not arise.

The axiomatisation of arithmetic is finite; the induction schema is represented by a single axiom with a universal quantification over a nominal unknown. This axiomatisation goes strictly further than previous axiomatisations in nominal algebra, making specific use of the first-order structure of PNL. It is also very close to the informal specification often given for first-order arithmetic.

To use a phrase that has become something of a slogan in nominal techniques, our axiomatisation of arithmetic, and we would argue more generally the natural use of PNL, is indeed 'ϵ away' from informal practice.

For the reader's convenience we now cut a tranche through existing nominal work, briefly sketching how the nominal ideas in this paper relate to what has come before. An extended discussion of related work is in the Conclusions.

Nominal techniques were introduced in [22], where the properties of Fraenkel-Mostowski set theory (**FM**) were studied.

Nominal logic (**NL**) [28] was another set of axioms in first-order logic; to use a bit of nominal jargon, it restricted to equivariant FM sets and removed the cumulative hierarchy. Both FM and NL were and remain important but note that the *language* used in both is just first-order terms and propositions.

A multi-level language (with atoms a and unknowns X) was introduced as nominal terms in [31]. The problem we

Supported by grant RYC-2006-002131 at the Polytechnic University of Madrid.
PPDP'10, July 26–28, 2010, Hagenberg, Austria.
Copyright © 2010 ACM 978-1-4503-0132-9/10/07. . . $10.00

$$\frac{a \in \mathbb{A}_\nu}{a : \nu} \qquad \frac{r_1 : \alpha_1 \ \ldots\ r_n : \alpha_n}{(r_1, \ldots, r_n) : (\alpha_1, \ldots, \alpha_n)} \qquad \frac{r : \alpha \quad (ar(\mathsf{f}) = (\alpha)\tau)}{\mathsf{f}(r) : \tau} \qquad \frac{r : \alpha\ (a \in \mathbb{A}_\nu)}{[a]r : [\nu]\alpha} \qquad \frac{(s(X) = \alpha)}{\pi{\cdot}X : \alpha}$$

$$\frac{}{\bot \text{ prop.}} \qquad \frac{\phi \text{ prop.} \ \ \psi \text{ prop.}}{\phi \Rightarrow \psi \text{ prop.}} \qquad \frac{r : \alpha \ \ (ar(\mathsf{P}) = \alpha)}{\mathsf{P}(r) \text{ prop.}} \qquad \frac{\phi \text{ prop.}}{\forall X.\phi \text{ prop.}}$$

Figure 1: Terms and propositions

see with this is that nominal terms have freshness contexts and cannot be conveniently quotiented by α-equivalence. We find this inconvenient for several reasons, one of which is difficulty introducing binders like $\forall X$ and λX.

We introduced permissive-nominal terms in [7]; these *can* be quotiented by α-equivalence and can be extended with binders for X. To a first approximation, this paper studies syntax, semantics, derivability, and expressivity of $\forall X$ in a permissive-nominal terms context. In another paper we make a similar study of λX.

2. Permissive-nominal Logic

2.1 Syntax

Definition 2.1. A **sort-signature** is a pair $(\mathcal{A}, \mathcal{B})$ of **name** and **base** sorts. ν will range over name sorts; τ will range over base sorts. A **sort language** is then inductively defined by

$$\alpha ::= \nu \mid \tau \mid (\alpha, \ldots, \alpha) \mid [\nu]\alpha.$$

Remark 2.2. Examples of base sorts are: 'λ-terms', 'formulae', 'π-calculus processes', and 'program environments', 'functions', 'truth-values', 'behaviours', and 'valuations'.

Examples of name sorts are 'variable symbols', 'channel names', or 'memory locations'.

Definition 2.3. A **term-signature** over a sort-signature $(\mathcal{A}, \mathcal{B})$ is a tuple $(\mathcal{F}, \mathcal{P}, ar)$ where:

- $\mathcal{F}$ and $\mathcal{P}$ are disjoint sets of **term-** and **proposition-formers**.
- ar assigns to each $\mathsf{f} \in \mathcal{F}$ a **term-former arity** $(\alpha)\tau$[1] and to each $\mathsf{P} \in \mathcal{P}$ a **proposition-former arity** α, where α and τ are in the sort-language determined by $(\mathcal{A}, \mathcal{B})$.

A **signature** $\mathcal{S}$ is then a tuple $(\mathcal{A}, \mathcal{B}, \mathcal{F}, \mathcal{P}, ar)$.

We write $\mathsf{f} : (\alpha)\tau$ for $ar(\mathsf{f}) = (\alpha)\tau$ and similarly we write $\mathsf{P} : \alpha$ for $ar(\mathsf{P}) = \alpha$.

Definition 2.4. For each τ fix two disjoint countably infinite set of **atoms** $\mathbb{A}^<_\tau$ and $\mathbb{A}^>_\tau$. Write $\mathbb{A}_\tau = \mathbb{A}^<_\tau \uplus \mathbb{A}^>_\tau$ ($\uplus$ denotes disjoint set union), and write:

$$\mathbb{A}^< = \bigcup \mathbb{A}^<_\tau \qquad \mathbb{A}^> = \bigcup \mathbb{A}^>_\tau \qquad \mathbb{A} = \bigcup \mathbb{A}_\tau$$

$a, b, c, \ldots$ will range over *distinct* atoms (we call this the **permutative** convention).

A **permission set** has the form $(\mathbb{A}^< \cup A) \setminus B$ where $A \subseteq \mathbb{A}^>$ and $B \subseteq \mathbb{A}^<$ are finite. S, T, and U will range over permissions sets.

The use of $\mathbb{A}^<$ and $\mathbb{A}^>$ ensures that permission sets are infinite and also co-infinite (their complement is also infinite).

Definition 2.5. A **(level 1) permutation** is a bijection π on atoms such that $a \in \mathbb{A}_\tau$ implies $\pi(a) \in \mathbb{A}_\tau$ always, and

$$nontriv(\pi) = \{a{\in}\mathbb{A} \mid \pi(a){\neq}a\}$$

is finite. π will range over permutations.

[1] We will write $((\alpha_1, \ldots, \alpha_n))\tau$ just as $(\alpha_1, \ldots, \alpha_n)\tau$.

Definition 2.6. For each signature $\mathcal{S} = (\mathcal{A}, \mathcal{B}, \mathcal{F}, \mathcal{P}, ar)$ and each sort α over $(\mathcal{A}, \mathcal{B})$ and permission set S fix a countably infinite set of **unknowns** of that sort and permission set. X, Y, Z will range over distinct unknowns. Write $s(X)$ for the sort and $p(X)$ for the permission set of X.

Permission sets correspond to the *freshness contexts* of [31]. Unlike freshness contexts they are fixed, and being co-infinite they guarantee infinitely many 'fresh names'. This gives a 'permissive' treatment of α-equivalence, as will become clear.

Definition 2.7. For each signature $\mathcal{S}$, define **terms** and **propositions** (over $\mathcal{S}$) inductively by the rules in Figure 1.

Remark 2.8. Our version of PNL has connectives $\bot$, $\Rightarrow$, and $\forall$. We could easily add other connectives like $\top$, $\wedge$, $\vee$, and $\exists$. Instead we treat them as a definable extension using the standard 'de Morgan' encoding.

We may write $id{\cdot}X$ just as X.

Example 2.9. Consider $\mathsf{lam}([b]\mathsf{app}(X, \mathsf{var}(b)))$ where $b \notin p(X)$; this represents the λ-term schema $\lambda y.(ty)$ where $y \notin fv(t)$.

app and lam are term-formers of arities $(\iota, \iota)\iota$ and $([\nu]\iota)\iota$. The sorts of b and X are ν (names) and ι (individuals) respectively.

2.2 Permissive-nominal logic syntax is finitary

Terms and propositions are finite trees. Permission sets differ finitely from $\mathbb{A}^<$ and so trivially admit a finite represenation.

In fact, permissive-nominal term α-equality can be decided in linear time. A permissive-nominal *unification* algorithm [7] has been implemented [25].

α-equivalence is defined in this paper in Definition 2.16.

2.3 Permutation actions

Nominal techniques suggest handling α-renaming using permutations. To a first approximation, if wherever the reader sees 'permutation action' they substitute 'α-renaming', then they will not go too far wrong.

Definition 2.10. Write $\pi \circ \pi'$ for functional composition (so $(\pi \circ \pi')(a) = \pi(\pi'(a))$), id for the identity permutation (so $id(a) = a$ always) and π^{-1} for inverse (so $\pi \circ \pi^{-1} = id$).

Define a (level 1) **permutation action** by:

$$\begin{array}{rlrl}
\pi{\cdot}a &\equiv \pi(a) & \pi{\cdot}(r_1, \ldots, r_n) &\equiv (\pi{\cdot}r_1, \ldots, \pi{\cdot}r_n) \\
\pi{\cdot}[a]r &\equiv [\pi(a)]\pi{\cdot}r & \pi{\cdot}(\pi'{\cdot}X) &\equiv (\pi{\circ}\pi'){\cdot}X \\
\pi{\cdot}\mathsf{f}(r) &\equiv \mathsf{f}(\pi{\cdot}r) & & \\
\pi{\cdot}\bot &\equiv \bot & \pi{\cdot}(\phi \Rightarrow \psi) &\equiv (\pi{\cdot}\phi) \Rightarrow (\pi{\cdot}\psi) \\
\pi{\cdot}\mathsf{P}(r) &\equiv \mathsf{P}(\pi{\cdot}r) & \pi{\cdot}(\forall X.\phi) &\equiv \forall X.\pi{\cdot}\phi
\end{array}$$

Definition 2.11. Let Π range over sort- and permission-set-preserving bijections on unknowns (so $s(\Pi(X)){=}s(X)$ and $p(\Pi(X)){=}p(X)$) such that $\{X \mid \Pi(X) \neq X\}$ is finite.

Write $\Pi \circ \Pi'$ for functional composition, Id for the identity permutation, and Π^{-1} for inverse, much as in Definition 2.10.

Define a (level 2) **permutation action** by:

$$\begin{array}{rclrcl} \Pi{\cdot}a &\equiv& a & \Pi{\cdot}(r_1,\dots,r_n) &\equiv& (\Pi{\cdot}r_1,\dots,\Pi{\cdot}r_n) \\ \Pi{\cdot}[a]r &\equiv& [a]\Pi{\cdot}r & \Pi{\cdot}(\pi{\cdot}X) &\equiv& \pi{\cdot}(\Pi(X)) \\ \Pi{\cdot}\mathsf{f}(r) &\equiv& \mathsf{f}(\Pi{\cdot}r) && & \\ \Pi{\cdot}\bot &\equiv& \bot & \Pi{\cdot}(\phi \Rightarrow \psi) &\equiv& (\Pi{\cdot}\phi) \Rightarrow (\Pi{\cdot}\psi) \\ \Pi{\cdot}\mathsf{P}(r) &\equiv& \mathsf{P}(\Pi{\cdot}r) & \Pi{\cdot}(\forall X.\phi) &\equiv& \forall \Pi(X).\Pi{\cdot}\phi \end{array}$$

2.4 Free level 1 and level 2 variables

Definition 2.12. Suppose A is a set of atoms and π is a level 1 permutation. Suppose U is a set of unknowns and Π is a level 2 permutation. Define $\pi{\cdot}A$ and $\Pi{\cdot}U$ by

$$\pi{\cdot}A = \{\pi(a) \mid a \in A\} \quad \text{and} \quad \Pi{\cdot}U = \{\Pi(X) \mid X \in U\}.$$

This is the standard **pointwise** permutation action on sets.

Definition 2.13. Define **free atoms** $fa(r)$ by:

$$\begin{array}{rclrcl} fa(\pi{\cdot}X) &=& \pi{\cdot}p(X) & fa([a]r) &=& fa(r) \setminus \{a\} \\ fa(\mathsf{f}(r)) &=& fa(r) & fa((r_1,\dots,r_n)) &=& \bigcup fa(r_i) \\ fa(a) &=& \{a\} &&& \end{array}$$

Define **free unknowns** $fV(r)$ and $fV(\phi)$ by:

$$\begin{array}{rclrcl} fV(a) &=& \varnothing & fV(\pi{\cdot}X) &=& \{X\} \\ fV([a]r) &=& fV(r) & fV((r_1,\dots,r_n)) &=& \bigcup fV(r_i) \\ fV(\mathsf{f}(r)) &=& fV(r) &&& \\ fV(\bot) &=& \varnothing & fV(\phi \Rightarrow \psi) &=& fV(\phi) \cup fV(\psi) \\ fV(\mathsf{P}(r)) &=& fV(r) & fV(\forall X.\phi) &=& fV(\phi) \setminus \{X\} \end{array}$$

Lemma 2.14. *$fa(\pi{\cdot}r) = \pi{\cdot}fa(r)$ and $fa(\pi{\cdot}\phi) = \pi{\cdot}fa(\phi)$.*
Also, $fV(\Pi{\cdot}r) = \Pi{\cdot}fV(r)$ and $fV(\Pi{\cdot}\phi) = \Pi{\cdot}fV(\phi)$.

Proof. By routine inductions on r. □

2.5 α-equivalence

Definition 2.15. Call a relation $\mathcal{R}$ on terms and on propositions a **congruence** when it is closed under the following rules:

$$\frac{r_i \mathrel{\mathcal{R}} s_i \quad 1 \le i \le n}{(r_1,\dots,r_n) \mathrel{\mathcal{R}} (s_1,\dots,s_n)} \qquad \frac{r \mathrel{\mathcal{R}} s \quad (\mathsf{f} : (\alpha)\tau,\ r,s : \alpha)}{\mathsf{f}(r) \mathrel{\mathcal{R}} \mathsf{f}(s)}$$

$$\frac{r \mathrel{\mathcal{R}} s}{[a]r \mathrel{\mathcal{R}} [a]s} \qquad \frac{\phi \mathrel{\mathcal{R}} \phi' \quad \psi \mathrel{\mathcal{R}} \psi'}{\phi \Rightarrow \psi \mathrel{\mathcal{R}} \phi' \Rightarrow \psi'}$$

$$\frac{r \mathrel{\mathcal{R}} s \quad (\mathsf{P} : \alpha,\ r,s : \alpha)}{\mathsf{P}(r) \mathrel{\mathcal{R}} \mathsf{P}(s)} \qquad \frac{\phi \mathrel{\mathcal{R}} \phi'}{\forall X.\phi \mathrel{\mathcal{R}} \forall X.\phi'}$$

Definition 2.16. Write $(a\ b)$ for the **(level 1) swapping** permutation which maps a to b, b to a, and all other c to themselves. Similarly write $(X\ Y)$ for the **(level 2) swapping**.

Define α**-equivalence** on terms and propositions to be the least congruence $=_\alpha$ such that:

$$\frac{(b\ a){\cdot}r =_\alpha s \quad (b \notin fa(r))}{[a]r =_\alpha [b]s} \qquad \frac{(\pi(a) = \pi'(a) \text{ all } a \in p(X))}{\pi{\cdot}X =_\alpha \pi'{\cdot}X}$$

$$\frac{(Y\ X){\cdot}\phi =_\alpha \psi \quad (Y \notin fV(\phi))}{\forall X.\phi =_\alpha \forall Y.\psi}$$

Example 2.17. We α-convert X and a in $\forall X.\mathsf{P}([a]X)$.

Let $s(Y) = s(X)$ and $p(Y) = p(X) = \mathbb{A}^<$. Suppose $b \notin \mathbb{A}^<$. Using $(a\ b)$ and $(X\ Y)$ we deduce:

$$\begin{array}{rcl} \forall X.\mathsf{P}([a]X) & \overset{(a\ b)}{=_\alpha} & \forall X.\mathsf{P}([b](b\ a){\cdot}X) \\ & \overset{(X\ Y)}{=_\alpha} & \forall Y.\mathsf{P}([b](b\ a){\cdot}Y). \end{array}$$

It is routine to convert this sketch into a full derivation-tree.

Remark 2.18. Note that α-equivalence is:

- Highly symmetric between levels 1 and 2.
- Based on permutations instead of substitutions (in keeping with the 'nominal' ideas in [22]).
- Does not require equality reasoning in the logic.

We emphasise this last point. We say that in PNL we can 'just α-convert'. We take this for granted in first- and higher-order logic. In this respect, PNL is close to informal practice.

In [4, 12, 14, 22, 28] it is not in general possible to 'just α-convert' a level 1 abstraction. We appeal instead to equality reasoning describing atoms-abstraction in nominal sets. But this is harder; derivable equality is more complex than syntactic equivalence.

Lemma 2.19. *For every π, Π, r, s, ϕ, and ψ, the following hold: $r =_\alpha s$ if and only if $\pi{\cdot}r =_\alpha \pi{\cdot}s$ and similarly $\phi =_\alpha \psi$ if and only if $\pi{\cdot}\phi =_\alpha \pi{\cdot}\psi$. Also, $r =_\alpha s$ if and only if $\Pi{\cdot}r =_\alpha \Pi{\cdot}s$, and similarly for $\phi =_\alpha \psi$.*

Lemma 2.20. *If $r =_\alpha s$ then $fa(r) = fa(s)$.*

Proposition 2.21. *$=_\alpha$ is an equivalence relation on terms and propositions.*

Proof. By a standard argument as in [10], using Lemmas 2.14, 2.20, and 2.19. □

Lemma 2.22. *If $\pi(a) = a$ for every $a \in fa(r)$ then $\pi{\cdot}r =_\alpha r$.*

PNL retains the 'nominal' power to reason on names and binding in nominal abstract syntax. See the example theories of freshness and inductive datatypes in Section 7.

2.6 Substitution

Definition 2.23. A (level 2) **substitution** θ is a function from unknowns to terms such that

- $\theta(X) : s(X)$ always,
- $fa(\theta(X)) \subseteq p(X)$ always,
- and $\theta(X) \equiv id{\cdot}X$ for all but finitely many X.

θ will range over substitutions. Define $nontriv(\theta)$ by:

$$nontriv(\theta) \equiv \{X \mid \theta(X) \not\equiv id{\cdot}X \text{ or } X \in fV(\theta(Y)) \text{ for some } Y\}$$

$nontriv(\theta)$ is unknowns that can be produced or consumed by θ, other than in the trivial manner that $\theta(X) \equiv id{\cdot}X$.

Definition 2.24. Define a **substitution action** by:

$$\begin{array}{rclrcl} a\theta &\equiv& a & (r_1,\dots,r_n)\theta &\equiv& (r_1\theta,\dots,r_n\theta) \\ ([a]r)\theta &\equiv& [a](r\theta) & (\pi{\cdot}X)\theta &\equiv& \pi{\cdot}\theta(X) \\ \mathsf{f}(r)\theta &\equiv& \mathsf{f}(r\theta) &&& \\ \bot\theta &\equiv& \bot & (\phi \Rightarrow \psi)\theta &\equiv& (\phi\theta) \Rightarrow \psi\theta \\ (\mathsf{P}(r))\theta &\equiv& \mathsf{P}(r\theta) & (\forall X.\phi)\theta &\equiv& \forall Y.(((Y\ X){\cdot}\phi)\theta) \end{array}$$

In the clause for $\forall X$ we rename X to be fresh for $nontriv(\theta)$, if necessary, using a fixed but arbitrary choice of fresh Y for each X, ϕ, θ.

Remark 2.25. Level 2 substitution $r\theta$ is capturing for level 1 abstraction $[a]$-. For example if $\theta(X) = a$ then $([a]X)\theta \equiv [a]a$. This is the behaviour displayed by the informal meta-level when we write "take t to be x in $\lambda x.t$".

Remark 2.26. Unknowns are variables; they have a substitution action. Atoms are not variables. Atoms are data; they are 'bindable constant symbols'.

$$\frac{}{\Phi,\,\phi\vdash\phi,\,\Psi}(\mathbf{Ax})\qquad \frac{}{\Phi,\,\bot\vdash\Psi}(\bot\mathbf{L})\qquad \frac{\Phi\vdash\phi,\,\Psi\quad \Phi,\,\psi\vdash\Psi}{\Phi,\,\phi\Rightarrow\psi\vdash\Psi}(\Rightarrow\mathbf{L})\qquad \frac{\Phi,\,\phi\vdash\psi,\,\Psi}{\Phi\vdash\phi\Rightarrow\psi,\,\Psi}(\Rightarrow\mathbf{R})\qquad \frac{\Phi,\,\phi\vdash\Psi}{\Phi,\,\pi\cdot\phi\vdash\Psi}(\text{И})$$

$$\frac{\begin{array}{c}\Phi,\,\phi[X{::=}r]\vdash\Psi\\ (fa(r){\subseteq}p(X),\ r{:}s(X))\end{array}}{\Phi,\,\forall X.\phi\vdash\Psi}(\forall\mathbf{L})\qquad \frac{\Phi\vdash\phi,\,\Psi\quad (X\notin fV(\Phi,\Psi))}{\Phi\vdash\forall X.\phi,\,\Psi}(\forall\mathbf{R})\qquad \frac{\Phi,\,\phi\vdash\Psi\quad (\phi=_\alpha\psi)}{\Phi,\,\psi\vdash\Psi}(\alpha_{\mathbf{L}})\qquad \frac{\Phi\vdash\phi,\,\Psi\quad (\phi=_\alpha\psi)}{\Phi\vdash\psi,\,\Psi}(\alpha_{\mathbf{R}})$$

Figure 2: Sequent derivation rules of Permissive-nominal Logic

The reader should not expect atoms to populate every sort, like variables do. Atoms populate their own special sorts, name-sorts, which are sorts for 'bindable data'.

We can make atoms populate a base sort (e.g. variable symbols with sort 'λ-terms' or 'functions') with a function-symbol (Definition 2.3), e.g. var in Sections 3 and 7.

We can make atoms behave like variables using axioms, like those of SUB in Figure 4. In PNL, a substitution action for atoms is a matter of writing suitable axioms. Fortunately, nominal techniques make this fairly easy to do.

2.7 Sequents and derivability

Definition 2.27. Φ and Ψ will range over sets of propositions. We may write ϕ,Φ and Φ,ϕ as shorthand for $\{\phi\}\cup\Phi$.

A **sequent** is a pair $\Phi\vdash\Psi$. Write

$$fV(\Phi,\Psi)=\bigcup\{fV(\phi)\mid\phi\in\Phi\}\cup\bigcup\{fV(\psi)\mid\psi\in\Psi\}.$$

The **derivable** sequents are inductively defined in Figure 2.

We may write $\Phi\vdash\Psi$ as shorthand for '$\Phi\vdash\Psi$ is a derivable sequent'. We may write $\Phi\nvdash\Psi$ as shorthand for '$\Phi\vdash\Psi$ is not a derivable sequent'.

Remark 2.28. Figure 2 includes rules for $\bot$, $\Rightarrow$, and $\forall$. We could add rules for other connectives like $\top$, $\wedge$, $\vee$, and $\exists$. Because the PNL in this paper is classical, we can treat derivation rules for them as a definable extension of what we already have.

We see no inherent difficulty with constructing an intuitionistic version of PNL.

Example 2.29. The condition $fa(r)\subseteq p(X)$ in ($\forall\mathbf{L}$) might suggest that $\forall X.\phi$ means "$\phi[X{::=}r]$ for every r such that $fa(r)\subseteq p(X)$". Indeed this is so, but this restriction is not quite what it seems.

Suppose a name sort Atm and suppose X : Atm and P : Atm. By ($\forall\mathbf{L}$), $\forall X.\mathsf{P}(X)\vdash\mathsf{P}(b)$ for some $b\in p(X)$. But then by considering the swapping $(a\ b)$ and (И), $\forall X.\mathsf{P}(X)\vdash\mathsf{P}(a)$ for *all* a, even if $a\notin p(X)$.

In spite of this $\forall X.\phi$ does not mean "$\phi[X{::=}r]$ for every r". This is because permutations are bijective. For example, suppose X : Atm, $a\notin p(X)$, and P : (Atm, Atm). Then $\forall X.\mathsf{P}(a,X)\vdash P(a,r)$ for all r such that $a\notin fa(r)$, and also $\forall X.\mathsf{P}(a,X)\vdash P(b,r)$ for all r and all b such that $b\notin fa(r)$.

In spite of this, $\forall X.\mathsf{P}(a,X)\nvdash P(a,a)$.

3. Arithmetic as a PNL theory

Definition 3.1. We start by defining the sorts, term-formers, and proposition-symbols of a signature $\dot{\mathcal{L}}$ which is suitable for finitely specifying arithmetic in PNL.

We assume one atomic sort ν and two base sorts ι and o. We assume term-formers

$0:\iota$	$\mathsf{succ}:(\iota)\iota$	$\dot{+}:(\iota,\iota)\iota$	$\dot{*}:(\iota,\iota)\iota$
$\dot{\bot}:o$	$\dot{\Rightarrow}:(o,o)o$	$\dot{\forall}:([\nu]o)o$	$\dot{\approx}:(\iota,\iota)o$
$\mathsf{var}:(\nu)\iota$	$\mathsf{sub}_\iota:([\nu]\iota,\iota)\iota$	$\mathsf{sub}_o:([\nu]o,\iota)o$	

and proposition-formers

$$\approx_\iota\!:(\iota,\iota)\qquad \approx_o\!:(o,o)\qquad \epsilon:(o).$$

We introduce the following syntactic sugar: we may write $\mathsf{sub}_o([a]r,t)$ as $r[a{\mapsto}t]$; we may write $\mathsf{sub}_\iota([a]r,t)$ as $r[a{\mapsto}t]$; and we may write both $\approx_\iota$ and $\approx_o$ just as $\approx$.

Example 3.2. Here are some example terms:

- $\dot{\forall}[a]X$. This represents the first-order logic schema $\forall x.\phi$. If $a\in p(X)$ then this corresponds to $x\in fv(\phi)$ being permitted; otherwise it is not. $\dot{\forall}$ is a function-symbol, of arity $([\iota]o)o$, and the sorts of a and X are ν (names) and o (truth-values) respectively.
- $\dot{\forall}[b](b\ a)\cdot X$ where $b\notin p(X)$. This also represents the first-order logic schema $\forall x.\phi$, but we have α-converted a to some fresh b. We might write this '$\forall y.\phi[y/x]$ where y is fresh'.

We now build up theories in PNL, bit by bit. Higher-order logic (e.g. in Isabelle/Pure) can be used in much the same way [26], but starting from a higher-order logical foundation. In a PNL implementation we would probably need just a few bells and whistles, e.g. polymorphism in the sorts.

Remark 3.3. Informally, we might call $\dot{\approx}$ 'object-level' equality, and $\approx$ 'meta-level' equality. Similarly $\dot{\bot}$ is 'object-level' and $\bot$ is 'meta-level'. ϵ converts 'object-level' truth to 'meta-level' truth. This is the same distinction as between = and ==, between $\forall$ and $\bigwedge$, and the same as the function from o to Prop, in Isabelle/FOL.

Perhaps *level 1* and *level 2* are a better terminology than 'object-level' and 'meta-level', since the above is all object-level in the sense that it is written formally in PNL and not informally in English.

Formally, $\dot{\forall}[a]\dot{\bot}$ is a term of type o, and $\forall X.\bot$ is a proposition. ϵ mediates between terms of type o and propositions.

Axioms for ...equality Axioms EQU for $\approx$: (ι,ι) and $\approx$: (o,o) are in Figure 3.

...substitution Axioms SUB for sub_ι and sub_o are in Figure 4.

We arguably abuse notation in Figure 4 when we use variables of sort ι and o as appropriate not necessarily giving them distinct names (e.g. in ($\mathbf{sub}{*}$) X has sort ι, whereas in ($\mathbf{sub}\dot{\Rightarrow}$) we use another variable also written X with sort o.

Remark 3.4. Note there are no axioms ($\mathbf{sub0}$) and ($\mathbf{sub}\dot{\bot}$). These are subsumed by ($\mathbf{sub}\#$). Note also that ($\mathbf{sub}\#$) really is necessary, for completeness; there might be models not all of whose elements are referenced by closed terms.

...first-order logic Axioms FOL for $\dot{\bot}$, $\dot{\Rightarrow}$, and $\dot{\forall}$ are in Figure 5.

...and arithmetic Given EQU, SUB, and FOL, it is not hard to write axioms for arithmetic in PNL. This is in Figure 6.

Remark 3.5. SUB is from [16], which includes a formal proof of soundness and completeness. In [17] first-order logic is equationally axiomatised using nominal algebra (so the axioms involve only equality). The axioms of FOL are *not* based on those of [17]. PNL has $\bot$, $\Rightarrow$, and $\forall$, whereas nominal algebra does not; we use this extra structure to capture the behaviour of $\dot{\bot}$, $\dot{\Rightarrow}$, and $\dot{\forall}$.

Remark 3.6. Instead of EQU we could extend Figure 2 with derivation rules as follows:

$$\frac{\Phi,\ r{\approx}s,\ \phi[X{::=}r],\ \phi[X{::=}s] \vdash \Psi \quad (fa(r){\cup}fa(s) \subseteq p(X))}{\Phi,\ r{\approx}s,\ \phi[X{::=}r] \vdash \Psi}\,(\approx\mathbf{S}) \qquad \frac{\Phi,\ r{\approx}r \vdash \Psi}{\Phi \vdash \Psi}\,(\approx\mathbf{R})$$

Remark 3.7. Every unknown has a sort, and a permission set. Different choices of permission set may yield logically equivalent results. For example, in (**sub**lam) it is not vital that $p(Z)$ is *exactly* $(b\ a)\cdot\mathbb{A}^<$. The important point is that $a \notin p(Z)$. Similarly, in (**sub**app) it is not vital that $p(X'') = p(X')$; when we use the axiom we can instantiate X'' and X' to r'' and r' such that $fa(r'') \neq fa(r')$, and conversely if we take $p(X'') \neq p(X')$ then we can still instantiate X'' and X' to r'' and r' such that $fa(r'') = fa(r') \subseteq p(X'')\cap p(X')$. More on this in Section 9.

As discussed in Example 2.29, these axioms apply to *all* instantiating terms.

Remark 3.8. These axioms conservatively extend the usual first-order logic axiomatisation of arithmetic in a sense we make formal and prove in Section 6. First, we must introduce a suitable notion of model.

4. Permissive-nominal sets

Nominal sets were introduced in [22]. (They were called 'equivariant FM sets', and given their modern name later in [28].) Nominal sets have a *finite support* property, that every element has a supporting finite set. Permissive-nominal sets have a similar property that every element has a supporting *permission* set (Definition 4.2). This need not be finite but most of the good properties of nominal sets are preserved.

Definition 4.1. A **set with a permutation action** X is a pair $(|\mathsf{X}|, \cdot)$ of

- a **carrier set** $|\mathsf{X}|$ and
- a group action on the carrier set $(Permutations \times |\mathsf{X}|) \to |\mathsf{X}|$, written infix as $\pi\cdot x$. (Here, we write $Permutations$ for the set of all finite permutations defined in Definition 2.5.) So, $id\cdot x = x$ and $\pi\cdot(\pi'\cdot x) = (\pi\circ\pi')\cdot x$ for every π and π' and every $x \in |\mathsf{X}|$.

Definition 4.2. Say $A \subseteq \mathbb{A}$ **supports** $x \in |\mathsf{X}|$ when for all permutations π, if $\pi(a) = a$ for all $a \in A$ then $\pi\cdot x = x$.

A **permissive-nominal set** is a set with a permutation action such that every element has supporting permission set. X, Y will range over permissive-nominal sets.

Theorem 4.3. *Every $x \in |\mathsf{X}|$ has a unique least supporting set $supp(x) \subseteq \mathbb{A}$.*

As a corollary, if $\pi(a) = a$ for all $a \in supp(x)$ then $\pi\cdot x = x$.

Proof. The corollary is immediate given the definition of support (Definition 4.2).

Define $S = \bigcap\{A \mid A$ permission set, supports $x\}$. Also, choose some permission set A that supports x.

Suppose $\pi(a) = a$ for all $a \in S$. Write $a_1,\dots,a_n$ for the atoms in $nontriv(\pi) \cap A$, in some order. Let $b_1,\dots,b_n$ be some choice of fresh atoms (so $b_i \notin A \cup nontriv(\pi) \cup S$ for $1 \leq i \leq n$). Write $\tau = (b_1\ a_1)\circ\ldots\circ(b_n\ a_n)$. It is routine to check that $(\tau\circ\pi\circ\tau)(a) = a$ for every $a \in A$. Thus $\tau\cdot(\pi\cdot(\tau\cdot x)) = x$. Now $\tau\cdot x = x$, and it follows by a routine manipulation that $\pi\cdot x = x$ as required. □

Lemma 4.4. *If $x \in |\mathsf{X}|$ then $supp(\pi\cdot x) = \pi\cdot supp(x)$ (Def. 2.12).*

Proof. By routine calculations using the group action. □

Corollary 4.5. *Suppose $x \in |\mathsf{X}|$ and $b \notin supp(x)$. Then $(b\ a)\cdot x = x$ implies $a \notin supp(x)$.*

Proof. Suppose $b \notin supp(x)$. We prove the contrapositive. Suppose $a \in supp(x)$. By Lemma 4.4 $supp((b\ a)\cdot x) = (b\ a)\cdot supp(x)$. By our suppositions, $(b\ a)\cdot supp(x) \neq supp(x)$. It follows that $(b\ a)\cdot x \neq x$. □

4.1 Examples of permissive-nominal sets: atoms; atoms-abstraction; cartesian product

Definition 4.6. $\mathbb{A}$ the set of atoms can be considered a nominal set with a natural permutation action $\pi\cdot a = \pi(a)$.

In the case of $\mathbb{A}$ only, we will be lax about the distinction between the set of atoms, and the permissive-nominal set of atoms with its natural permutation action.

Definition 4.7. Suppose $x \in |\mathsf{X}|$ and $a \in \mathbb{A}_\tau$. Define:

$$[a]x = \{(a,x)\} \cup \{(b, (b\ a)\cdot x \mid b \in \mathbb{A}_\tau \setminus supp(x)\}$$
$$|[\mathbb{A}_\tau]\mathsf{X}| = [\mathbb{A}_\tau]\mathsf{X} = \{[a]x \mid a \in \mathbb{A}_\tau,\ x \in |\mathsf{X}|\}$$

Give $[\mathbb{A}_\tau]\mathsf{X}$ the permutation action $\pi\cdot[a]x = [\pi(a)]\pi\cdot x$.

Lemma 4.8.
- *$[\mathbb{A}_\tau]\mathsf{X}$ is a permissive-nominal set.*
- *$[a]x{=}[a]x'$ if and only if $x{=}x'$, for $a{\in}\mathbb{A}_\tau$ and $x{\in}|\mathsf{X}|$.*
- *$[a]x{=}[a']x'$ if and only if $a'{\notin}supp(x)$ and $(a'\ a)\cdot x{=}x'$, for $a, a'{\in}\mathbb{A}_\tau$ and $x, x'{\in}|\mathsf{X}|$.*

Definition 4.9. If X_i are nominal sets for $1 \leq i \leq n$ then define $\mathsf{X}_1 \times \ldots \times \mathsf{X}_n$ by:

$$|\mathsf{X}_1 \times \ldots \times \mathsf{X}_n| = |\mathsf{X}_1| \times \ldots \times |\mathsf{X}_n|$$
$$\pi\cdot(x_1,\dots,x_n) = (\pi\cdot x_1,\dots,\pi\cdot x_n)$$

Lemma 4.10.
- *$supp(a) = \{a\}$.*
- *$supp([a]x) = supp(x) \setminus \{a\}$.*
- *$supp((x_1,\dots,x_n)) = \bigcup\{supp(x_i) \mid 1 \leq i \leq n\}$.*

Proof. Proofs are as in [22]. □

Definition 4.11. Suppose X and Y are sets with a permutation action. Call a function f from $|\mathsf{X}|$ to $|\mathsf{Y}|$ **finite equivariant** when $\pi\cdot(f(x)) = f(\pi\cdot x)$ for all $x \in |\mathsf{X}|$.

Call a subset $U \subseteq |\mathsf{X}|$ **finite equivariant** when $x \in U$ if and only if $\pi\cdot x \in U$ for all $x \in |\mathsf{X}|$ and all π (so $\pi\cdot U = U$ always, extending the permutation action pointwise to sets).

Remark 4.12. We write '*finite*' equivariant, because it is possible to be equivariant up to finite permutations but not up to infinite permutations. The distinction was first observed as 'fuzzy support' in [13]. More on this in a later paper.

$$\begin{array}{rlr}
\forall X',X,Y',Y.(X'\approx X \wedge Y'\approx Y) & \Rightarrow X'\ op\ Y' \approx X\ op\ Y & op\in\{+,*,\dot{\Rightarrow},\dot{\approx}\}\\
\forall X',X. & X'\approx X \Rightarrow \mathsf{succ}(X')\approx \mathsf{succ}(X) & \\
 & op \approx op & op\in\{0,\dot{\bot}\}\\
\forall Z',Z. & Z'\approx Z \Rightarrow \dot{\forall}([a]Z')\approx \dot{\forall}([a]Z) & \\
\forall X',X,Y',Y.(X'\approx X \wedge Y'\approx Y) & \Rightarrow op([a]X',Y')\approx op([a]X,Y) & op\in\{\mathsf{sub}_\iota,\mathsf{sub}_o\}\\
\forall Z',Z. & Z'\approx Z \Rightarrow (\epsilon(Z')\Leftrightarrow \epsilon(Z)) &
\end{array}$$

We fill in sorts as appropriate. Thus, $\dot{\bot}\approx_o\dot{\bot}$ whereas $0\approx_\iota 0$, and so on.

Figure 3: EQU: axioms for equality as a PNL theory

$$\begin{array}{llll}
(\mathbf{subvar}) & \forall X.\ \mathsf{var}(a)[a\mapsto X] & \approx X & \\
(\mathbf{sub\#}) & \forall X,Z.\ Z[a\mapsto X] & \approx Z & (p(Z)=(b\ a)\cdot\mathbb{A}^<)\\
(\mathbf{subsucc}) & \forall X',X.\ \mathsf{succ}(X')[a\mapsto X] & \approx \mathsf{succ}(X'[a\mapsto X]) & \\
(\mathbf{sub}op) & \forall X'',X',X.\ (X''\ op\ X')[a\mapsto X] & \approx (X''[a\mapsto X]\ op\ X'[a\mapsto X]) & (op\in\{+,*,\dot{\Rightarrow},\dot{\approx}\})\\
(\mathbf{sub}\dot{\forall}) & \forall X,Z.\ (\dot{\forall}([b]Z))[a\mapsto X] & \approx \dot{\forall}([b](Z[a\mapsto X])) & (p(Z)=(b\ a)\cdot\mathbb{A}^<)\\
(\mathbf{subid}) & \forall X.\ X[a\mapsto \mathsf{var}(a)] & \approx X &
\end{array}$$

$a\in\mathbb{A}^<$ and $b\notin\mathbb{A}^<$. The permission set of X'', X', and X is equal to $\mathbb{A}^<$. The permission set of Z is equal to $(b\ a)\cdot\mathbb{A}^<$.

Figure 4: SUB: axioms for substitution as a PNL theory

$$\begin{array}{lll}
(\dot{\Rightarrow}) & \forall Z',Z.\ \epsilon(Z'\dot{\Rightarrow}Z) & \Leftrightarrow (\epsilon(Z')\Rightarrow\epsilon(Z))\\
(\dot{\forall}) & \forall Z.\ \big(\epsilon(\dot{\forall}([a]Z)) & \Leftrightarrow \forall X.\epsilon(Z[a\mapsto X])\big)\\
(\dot{\bot}) & \epsilon(\dot{\bot}) & \Rightarrow \bot\\
(\dot{\approx}) & \forall X',X.\ X'\approx X & \Rightarrow \epsilon(X'\dot{\approx}X)
\end{array}$$

Here Z' and Z have sort o and permission set $\mathbb{A}^<$; X' and X have sort ι and permission set $\mathbb{A}^<$; and $a\in\mathbb{A}^<$.

Figure 5: FOL: axioms for first-order logic as a PNL theory

$$\begin{array}{llll}
(\mathbf{PS0}) & \forall X.\ \mathsf{succ}(X)\approx 0\Rightarrow\bot & (\mathbf{P*0}) & \forall X.\ X*0\approx 0\\
(\mathbf{PSS}) & \forall X',X.\ \mathsf{succ}(X')\approx\mathsf{succ}(X)\Rightarrow X'\approx X & (\mathbf{P*succ}) & \forall X',X.\ X'*\mathsf{succ}(X)\approx(X'*X)+X\\
(\mathbf{P+0}) & \forall X.\ X+0\approx X & (\mathbf{PInd}) & \forall Z.\ (\epsilon(Z[a\mapsto 0])\Rightarrow\\
(\mathbf{P+succ}) & \forall X',X.\ X'+\mathsf{succ}(X)\approx\mathsf{succ}(X')+X & & \quad\big(\forall X.(\epsilon(Z[a\mapsto X])\Rightarrow\epsilon(Z[a\mapsto\mathsf{succ}(X)]))\big)\Rightarrow\\
 & & & \quad\forall X.\epsilon(Z[a\mapsto X]))
\end{array}$$

All variables have permission set $\mathbb{A}^<$, and $a\in\mathbb{A}^<$.

Figure 6: ARITH: axioms for arithmetic as a PNL theory

5. Semantics

We now interpret PNL in a notion of permissive-nominal set. We sketch a proof of soundness (Theorem 5.15). PNL is also complete for the same semantics; more on that in a journal version of this paper.

Definition 5.1. Suppose $(\mathcal{A},\mathcal{B})$ is a sort-signature (Def. 2.1).

An **interpretation** $\mathcal{I}$ for $(\mathcal{A},\mathcal{B})$ consists of an assignment of a permissive-nominal set $\tau^{\mathcal{I}}$ to each $\tau\in\mathcal{B}$.

We extend $\mathcal{I}$ to sorts by:

$$\begin{array}{ll}
[\![\tau]\!]^{\mathcal{I}}=\tau^{\mathcal{I}} & [\![(\alpha_1,\dots,\alpha_n)]\!]^{\mathcal{I}}=[\![\alpha_1]\!]^{\mathcal{I}}\times\ldots\times[\![\alpha_n]\!]^{\mathcal{I}}\\
[\![\nu]\!]^{\mathcal{I}}=\mathbb{A}_\nu & [\![[\nu]\alpha]\!]^{\mathcal{I}}=[\mathbb{A}_\nu][\![\alpha]\!]^{\mathcal{I}}
\end{array}$$

Definition 5.2. Suppose $\mathcal{S}=(\mathcal{A},\mathcal{B},\mathcal{F},\mathcal{P},ar)$ is a signature (Definition 2.3).

An **interpretation** $\mathcal{I}$ for $\mathcal{S}$ consists of the following data:

- An interpretation for the sort-signature $(\mathcal{A},\mathcal{B})$ (Def. 5.1).
- For every $\mathsf{f}\in\mathcal{F}$ with $ar(\mathsf{f})=(\alpha)\tau$ a finite equivariant function $\mathsf{f}^{\mathcal{I}}$ from $(\alpha)^{\mathcal{I}}$ to $(\tau)^{\mathcal{I}}$ (Definition 4.11).
- For every $\mathsf{P}\in\mathcal{P}$ with $ar(\mathsf{P})=\alpha$ a finite equivariant function $\mathsf{P}^{\mathcal{I}}$ from $(\alpha)^{\mathcal{I}}$ to $\{0,1\}$.

Definition 5.3. Suppose $\mathcal{I}$ is an interpretation for $\mathcal{S}$. A **valuation** ς to $\mathcal{I}$ is a map on unknowns such that for each X,

- $\varsigma(X)\in[\![s(X)]\!]^{\mathcal{I}}$, and
- $supp(\varsigma(X))\subseteq p(X)$.

ς will range over valuations.

Definition 5.4. Suppose $\mathcal{I}$ is an interpretation of a signature $\mathcal{S}$. Suppose ς is a valuation to $\mathcal{I}$.

Define an **interpretation** $[\![r]\!]^{\mathcal{I}}_\varsigma$ in $\mathcal{S}$ inductively by:

$$\begin{array}{rlrl}
[\![a]\!]^{\mathcal{I}}_\varsigma & =a & [\![[a]r]\!]^{\mathcal{I}}_\varsigma & =[a][\![r]\!]^{\mathcal{I}}_\varsigma\\
[\![\mathsf{f}(r)]\!]^{\mathcal{I}}_\varsigma & =\mathsf{f}^{\mathcal{I}}([\![r]\!]^{\mathcal{I}}_\varsigma) & [\![\pi\cdot X]\!]^{\mathcal{I}}_\varsigma & =\pi\cdot\varsigma(X)\\
[\![(r_1,\dots,r_n)]\!]^{\mathcal{I}}_\varsigma & =([\![r_1]\!]^{\mathcal{I}}_\varsigma,\dots,[\![r_n]\!]^{\mathcal{I}}_\varsigma) & &
\end{array}$$

Lemma 5.5. *If $r:\alpha$ then $[\![r]\!]^{\mathcal{I}}_\varsigma\in[\![\alpha]\!]^{\mathcal{I}}$.*

Lemma 5.6. $\pi\cdot[\![r]\!]^{\mathcal{I}}_\varsigma=[\![\pi\cdot r]\!]^{\mathcal{I}}_\varsigma$.

Proof. By a routine induction on r. We consider one case:

- The case $\pi'\cdot X$. By Definition 5.4 $[\![\pi'\cdot X]\!]^{\mathcal{I}}_\varsigma=\pi'\cdot\varsigma(X)$. Therefore $\pi\cdot[\![\pi'\cdot X]\!]^{\mathcal{I}}_\varsigma=\pi\cdot(\pi'\cdot\varsigma(X))$. It is a fact of the group action (Definition 4.1) that $\pi\cdot(\pi'\cdot\varsigma(X))=(\pi\circ$

$\pi')\cdot\varsigma(X)$, and of the permutation action (Definition 2.10) that $\pi\cdot(\pi'\cdot X) \equiv (\pi\circ\pi')\cdot X$. The result follows. □

Lemma 5.7. *$supp(\llbracket r\rrbracket_\varsigma) \subseteq fa(r)$.*

Proof. Routine induction using Lemmas 4.4 and 4.10. □

Lemma 5.8. *If $r =_\alpha s$ then for all ς, $\llbracket r\rrbracket^{\mathcal{I}}_\varsigma = \llbracket s\rrbracket^{\mathcal{I}}_\varsigma$.*

Proof. The non-trivial part is to check that if $a \notin fa(r)$ and $b \notin fa(r)$ then $\llbracket(a\ b)\cdot r\rrbracket^{\mathcal{I}}_\varsigma = \llbracket r\rrbracket^{\mathcal{I}}_\varsigma$. Suppose $a \notin fa(r)$ and $b \notin fa(r)$. By Lemma 5.7 $a \notin supp(\llbracket r\rrbracket_\varsigma)$ and $b \notin supp(\llbracket r\rrbracket_\varsigma)$. By the definition of support in Definition 4.2, $(a\ b)\cdot\llbracket r\rrbracket^{\mathcal{I}}_\varsigma = \llbracket r\rrbracket^{\mathcal{I}}_\varsigma$. We use Lemma 5.6. □

Definition 5.9. Suppose ς is a valuation to an interpretation $\mathcal{I}$. Suppose X is an unknown and $x \in (s(X))^{\mathcal{I}}$ is such that $supp(x) \subseteq p(X)$. Define a function $\varsigma[X{::=}x]$ by

$$(\varsigma[X{::=}x])(Y) = \varsigma(Y) \quad \text{and} \quad (\varsigma[X{::=}x])(X) = x.$$

It is easy to verify that $\varsigma[X{::=}x]$ is also a valuation to $\mathcal{I}$.

Definition 5.10. If Q is a set of numbers write $min\,Q$ for the least and $max\,Q$ for the greatest elements in Q, as is standard.

Definition 5.11. Suppose that $\mathcal{I}$ is an interpretation. Define an **interpretation of propositions** inductively by:

$$\begin{aligned}
\llbracket \mathsf{P}(r)\rrbracket^{\mathcal{I}}_\varsigma &= \mathsf{P}^{\mathcal{I}}(\llbracket r\rrbracket^{\mathcal{I}}_\varsigma)\\
\llbracket \bot\rrbracket^{\mathcal{I}}_\varsigma &= 0\\
\llbracket \phi\Rightarrow\psi\rrbracket^{\mathcal{I}}_\varsigma &= max\{1-\llbracket\phi\rrbracket^{\mathcal{I}}_\varsigma,\ \llbracket\psi\rrbracket^{\mathcal{I}}_\varsigma\}\\
\llbracket \forall X.\phi\rrbracket^{\mathcal{I}}_\varsigma &= min\{\llbracket\phi\rrbracket^{\mathcal{I}}_{\varsigma[X::=x]} \mid x{\in}\llbracket s(X)\rrbracket^{\mathcal{I}},\ supp(x){\subseteq}p(X)\}
\end{aligned}$$

Lemma 5.12. *$\llbracket\phi\rrbracket^{\mathcal{I}}_\varsigma = \llbracket\pi\cdot\phi\rrbracket^{\mathcal{I}}_\varsigma$ always.*

Proof. By induction on ϕ. We consider two cases:

- The case $\forall X.\phi$. Suppose $\llbracket\forall X.\phi\rrbracket^{\mathcal{I}}_\varsigma = 1$. This means that $\llbracket\phi\rrbracket^{\mathcal{I}}_{\varsigma[X::=x]} = 1$ for all $x \in \llbracket\alpha\rrbracket^{\mathcal{I}}$ such that $supp(x) \subseteq p(X)$. By inductive hypothesis $\llbracket\pi\cdot\phi\rrbracket^{\mathcal{I}}_{\varsigma[X::=x]} = 1$ for all $x \in \llbracket\alpha\rrbracket^{\mathcal{I}}$ such that $supp(x) \subseteq p(X)$. Therefore $\llbracket\forall X.\pi\cdot\phi\rrbracket^{\mathcal{I}}_\varsigma = 1$. The result follows, since $\pi\cdot(\forall X.\phi) \equiv \forall X.\pi\cdot\phi$.
- The case $\mathsf{P}(r)$. Suppose $\llbracket\mathsf{P}(r)\rrbracket^{\mathcal{I}}_\varsigma = 1$, so $\mathsf{P}^{\mathcal{I}}(\llbracket r\rrbracket^{\mathcal{I}}_\varsigma) = 1$. By assumption $\mathsf{P}^{\mathcal{I}}$ is an equivariant function, and it follows that $\mathsf{P}^{\mathcal{I}}(\pi\cdot\llbracket r\rrbracket^{\mathcal{I}}_\varsigma) = 1$. By Lemma 5.6 $\pi\cdot\llbracket r\rrbracket^{\mathcal{I}}_\varsigma = \llbracket\pi\cdot r\rrbracket^{\mathcal{I}}_\varsigma$. Thus $\mathsf{P}^{\mathcal{I}}(\llbracket\pi\cdot r\rrbracket^{\mathcal{I}}_\varsigma) = 1$, and so $\llbracket\pi\cdot\mathsf{P}(r)\rrbracket^{\mathcal{I}}_\varsigma = 1$. □

Lemma 5.13. *If $\phi =_\alpha \psi$ then $\llbracket\phi\rrbracket^{\mathcal{I}}_\varsigma = \llbracket\psi\rrbracket^{\mathcal{I}}_\varsigma$.*

Proof. By routine calculations using Lemma 5.8 for α-equality of terms in propositions. □

Definition 5.14 (Validity)**.** Call the proposition ϕ **valid** in $\mathcal{I}$ when $\llbracket\phi\rrbracket^{\mathcal{I}}_\varsigma = 1$ for all ς. Call the sequent $\phi_1,\ldots,\phi_n \vdash \psi_1,\ldots,\psi_p$ **valid** in $\mathcal{I}$ when $(\phi_1\wedge\ldots\wedge\phi_n)\Rightarrow(\psi_1\vee\ldots\vee\psi_p)$ is valid.

Theorem 5.15 (Soundness)**.** *If $\Phi\vdash\Psi$ is derivable then it is valid in all interpretations.*

Proof. By induction on the derivation of $\Phi\vdash\Psi$ (Figure 2). The case of (И) uses Lemma 5.12. The cases of $(\alpha_{\mathbf{L}})$ and $(\alpha_{\mathbf{R}})$ use Lemma 5.13. Other rules are routine by unpacking definitions. □

6. Arithmetic

6.1 A language $\mathcal{L}$ for first-order logic

For this section, $a, b, c, \ldots$ range over distinct atoms in $\mathbb{A}_\nu$, where $\mathbb{A}_\nu$ is the set of atoms used in $\dot{\mathcal{L}}$ in Section 3 (this is not necessary, but it is convenient).

Definition 6.1. Define **terms** and **sentences** of $\mathcal{L}$ by:

$$\begin{aligned}
t &::= a \mid 0 \mid succ(t) \mid t+t \mid t*t\\
\xi &::= t\approx t \mid \bot \mid \xi\Rightarrow\xi \mid \forall a.\xi
\end{aligned}$$

Substitution $t'[a{::=}t]$ and $\xi[a{::=}t]$ is as usual for first-order logic. We write sequents $\Xi\vdash\mathcal{X}$ where Ξ and $\mathcal{X}$ are sets of sentences. Derivability is as usual for first-order logic.

Definition 6.2. Define a mapping $(\text{-})^{\boldsymbol{\cdot}}$ from terms and sentences of $\mathcal{L}$ to terms of $\dot{\mathcal{L}}$ inductively as follows:

$$\begin{aligned}
(a)^{\boldsymbol{\cdot}} &= a & (0)^{\boldsymbol{\cdot}} &= 0\\
(succ(t))^{\boldsymbol{\cdot}} &= \mathsf{succ}((t)^{\boldsymbol{\cdot}}) & (t'+t)^{\boldsymbol{\cdot}} &= (t')^{\boldsymbol{\cdot}}+(t)^{\boldsymbol{\cdot}}\\
(t'*t)^{\boldsymbol{\cdot}} &= (t')^{\boldsymbol{\cdot}}*(t)^{\boldsymbol{\cdot}} & &\\
(t'\approx t)^{\boldsymbol{\cdot}} &= (t')^{\boldsymbol{\cdot}}\dot{\approx}(t)^{\boldsymbol{\cdot}} & (\bot)^{\boldsymbol{\cdot}} &= \dot{\bot}\\
(\xi'\Rightarrow\xi)^{\boldsymbol{\cdot}} &= (\xi')^{\boldsymbol{\cdot}}\mathrel{\dot{\Rightarrow}}(\xi)^{\boldsymbol{\cdot}} & (\forall a.\xi)^{\boldsymbol{\cdot}} &= \dot{\forall}[a](\xi)^{\boldsymbol{\cdot}}
\end{aligned}$$

Definition 6.3. Extend $(\text{-})^{\boldsymbol{\cdot}}$ to first-order logic sequents $\Xi\vdash\mathcal{X}$ as follows:

$$(\Xi\vdash\mathcal{X})^{\boldsymbol{\cdot}} = \epsilon(\dot{\forall}[a_1]\ldots\dot{\forall}[a_n]((\xi_1\wedge\ldots\wedge\xi_k)\Rightarrow(\chi_1\vee\ldots\vee\chi_l))^{\boldsymbol{\cdot}})$$

Here, $\Xi = \{\xi_1,\ldots,\xi_k\}$, $\mathcal{X} = \{\chi_1,\ldots,\chi_l\}$, and the free variables of Ξ and $\mathcal{X}$ are $\{a_1,\ldots,a_n\}$ (in some order).

Notation 6.4. Write $\mathcal{S}$ for $\mathsf{EQU}\cup\mathsf{SUB}\cup\mathsf{FOL}$.

Lemma 6.5. $\mathcal{S}\vdash(t'[a{::=}t])^{\boldsymbol{\cdot}}\approx(t')^{\boldsymbol{\cdot}}[a{\mapsto}(t)^{\boldsymbol{\cdot}}]$ *and*
$\mathcal{S}\vdash(\xi[a{::=}t])^{\boldsymbol{\cdot}}\approx(\xi)^{\boldsymbol{\cdot}}[a{\mapsto}(t)^{\boldsymbol{\cdot}}]$.

Proof. By routine inductions on t and ξ. □

Theorem 6.6 (Correctness)**.** *If $\Xi\vdash\mathcal{X}$ is derivable in first-order logic then $\mathcal{S}\vdash(\Xi\vdash\mathcal{X})^{\boldsymbol{\cdot}}$ is derivable in PNL.*

Proof. By a long but routine inspection we can check that EQU, SUB, and FOL allow us to model the behaviour of 'real' first-order logic. We use Lemma 6.5. □

6.2 Interpretation of first-order logic

We recall the usual definition of interpretations in first-order logic:

Definition 6.7. A **(first-order logic) interpretation** $\mathcal{M}$ is

- a **carrier set** M, and
- elements $0^{\mathcal{M}} \in M$, $succ^{\mathcal{M}} \in M\to M$, $+^{\mathcal{M}} \in (M\times M)\to M$, and $*^{\mathcal{M}} \in (M\times M)\to M$.

It is convenient to fix some $\mathcal{M}$ from here until Theorem 6.20.

Definition 6.8. Define $Valu_{\mathbb{A}_\nu}(M)$ by:

$$\begin{aligned}
Valu_{\mathbb{A}_\nu}(M) = \{\varepsilon\in\mathbb{A}_\nu\to M \mid \exists A\subseteq\mathbb{A}_\nu.A \text{ finite}\wedge\\
\forall a,b\notin A.\varepsilon(a)=\varepsilon(b)\}
\end{aligned} \quad (1)$$

Call elements of $Valu_{\mathbb{A}_\nu}(M)$ **$\mathbb{A}_\nu$-valuations** (to M).

ε will range over $\mathbb{A}_\nu$-valuations.

If $x\in M$ write $\varepsilon[a{::=}x]$ for the valuation mapping b to $\varepsilon(b)$ and mapping a to x.

Give ε a permutation action defined by $(\pi\cdot\varepsilon)(a) = \varepsilon(\pi^{-1}(a))$. Give subsets X of $Valu_{\mathbb{A}_\nu}(M)$ the **pointwise** permutation action defined by $\pi\cdot X = \{\pi\cdot\varepsilon \mid \varepsilon\in X\}$.

$(\mathbf{ps0})$	$\forall a.\ succ(a) \approx 0 \Rightarrow \bot$	$(\mathbf{p{*}0})$	$\forall a.\ a * 0 \approx 0$
$(\mathbf{pss})$	$\forall a', a.\ succ(a) \approx succ(a') \Rightarrow a \approx a'$	$(\mathbf{p{*}succ})$	$\forall a', a.\ a' * succ(a) \approx (a' * a) + a$
$(\mathbf{p{+}0})$	$\forall a.\ a + 0 \approx a$	$(\mathbf{pind})$	$\xi[a{::=}0] \Rightarrow \forall a.(\xi \Rightarrow \xi[a{::=}succ(a)]) \Rightarrow \forall a.\xi$ (every ξ, every a)
$(\mathbf{p{+}succ})$	$\forall a', a.\ a' + succ(a) \approx succ(a') + a$		

Figure 7: arithmetic: axioms for arithmetic in first-order logic

U, V will range over *finitely-supported* subsets of $Valu_{\mathbb{A}_\nu}(M)$ — so there exists some finite $A \subseteq \mathbb{A}_\nu$ such that for all π, if $\pi(a) = a$ for all $a \in A$ then $\pi{\cdot}U = U$.

Remark 6.9. $Valu_{\mathbb{A}_\nu}(M)$ would normally just be called 'the set of valuations'. We are more specific because we already have a notion of valuation on unknowns X (Definition 5.3).

PNL atoms are serving as variable symbols of $\mathcal{L}$. To conveniently apply nominal techniques, it is useful to restrict to valuations that are finite in the sense given in (1). In any case, any term or sentence will only contain finitely many atoms.

Definition 6.10. We extend the interpretation to first-order logic syntax as follows:

$$\begin{aligned}
\llbracket a\rrbracket^{\mathcal{M}}_{\varepsilon} &= \varepsilon(a)\\
\llbracket 0\rrbracket^{\mathcal{M}}_{\varepsilon} &= 0^{\mathcal{M}}\\
\llbracket succ(t)\rrbracket^{\mathcal{M}}_{\varepsilon} &= succ^{\mathcal{M}}(\llbracket t\rrbracket^{\mathcal{M}}_{\varepsilon})\\
\llbracket t' + t\rrbracket^{\mathcal{M}}_{\varepsilon} &= +^{\mathcal{M}}(\llbracket t'\rrbracket^{\mathcal{M}}_{\varepsilon}, \llbracket t\rrbracket^{\mathcal{M}}_{\varepsilon})\\
\llbracket t' * t\rrbracket^{\mathcal{M}}_{\varepsilon} &= *^{\mathcal{M}}(\llbracket t'\rrbracket^{\mathcal{M}}_{\varepsilon}, \llbracket t\rrbracket^{\mathcal{M}}_{\varepsilon})\\
\llbracket \bot\rrbracket^{\mathcal{M}}_{\varepsilon} &= 0\\
\llbracket \xi' \Rightarrow \xi\rrbracket^{\mathcal{M}}_{\varepsilon} &= max\{1{-}\llbracket \xi'\rrbracket^{\mathcal{M}}_{\varepsilon}, \llbracket \xi\rrbracket^{\mathcal{M}}_{\varepsilon}\}\\
\llbracket \forall a.\xi\rrbracket^{\mathcal{M}}_{\varepsilon} &= min\{\llbracket \xi\rrbracket^{\mathcal{M}}_{\varepsilon[a::=x]} \mid x \in M\}\\
\llbracket t' \approx t\rrbracket^{\mathcal{M}}_{\varepsilon} &= 1 \text{ if } \llbracket t'\rrbracket^{\mathcal{M}}_{\varepsilon} = \llbracket t\rrbracket^{\mathcal{M}}_{\varepsilon} \text{ and } 0 \text{ otherwise}
\end{aligned}$$

Definition 6.11. Call the sentence ξ **valid** in $\mathcal{M}$ when $\llbracket \xi\rrbracket^{\mathcal{M}}_{\varepsilon} = 1$ for all ε.

Call $\xi_1, \ldots, \xi_k \vdash \chi_1, \ldots, \chi_l$ **valid** in $\mathcal{M}$ when $(\xi_1 \wedge \ldots \wedge \xi_k) \Rightarrow (\chi_1 \vee \ldots \vee \chi_l)$ is valid.

6.3 Axioms and models

Definition 6.12. Define a first-order theory of **arithmetic** by the axioms in Figure 7.

An interpretation is a **model** of arithmetic when $\llbracket \xi\rrbracket^{\mathcal{M}} = 1$ for ξ each of $(\mathbf{ps0})$, $(\mathbf{pss})$, $(\mathbf{p{+}0})$, $(\mathbf{p{+}succ})$, $(\mathbf{p{*}0})$, $(\mathbf{p{*}succ})$, and every instance of $(\mathbf{pind})$.

Remark 6.13. $(\mathbf{pind})$, the induction axiom-scheme, is of course of particular interest. We therefore unpack what its validity

$$\llbracket \xi[a{::=}0] \Rightarrow \forall a.(\xi \Rightarrow \xi[a{::=}succ(a)]) \Rightarrow \forall a.\xi\rrbracket^{\mathcal{M}} = 1$$
$$(\text{every } \xi, \text{every } a)$$

means, in a little more detail. For every a and ξ:

- If $\llbracket \xi[a{::=}0]\rrbracket^{\mathcal{M}}_{\varepsilon} = 1$, and
- if for every $x \in M$, $\llbracket \xi\rrbracket^{\mathcal{M}}_{\varepsilon[a::=x]} = 1$ implies that $\llbracket \xi[a{::=}succ(a)]\rrbracket^{\mathcal{M}}_{\varepsilon[a::=x]} = 1$
- then for every $x \in M$, $\llbracket \xi\rrbracket^{\mathcal{M}}_{\varepsilon[a::=x]} = 1$.

In $(\mathbf{pind})$ we take 'every a', and in $(\mathbf{PInd})$ we do not. This is because in $(\mathbf{PInd})$, a is α-convertible,

6.4 Building an interpretation for $\dot{\mathcal{L}}$ out of one for $\mathcal{L}$

Recall the PNL signature $\dot{\mathcal{L}}$ from Section 3. Suppose $\mathcal{M}$ is a model of $\mathcal{L}$. We use it to build an interpretation $\mathcal{N}$ of $\dot{\mathcal{L}}$.

Definition 6.14. Extend $\mathcal{L}$ to $\mathcal{L}{+}M$ where we add elements of M as constants, and extend the interpretation to interpret these constants as themselves in M. (So if $x \in M$ then x is a constant symbol in $\mathcal{L}{+}M$ and $\llbracket x\rrbracket^{\mathcal{M}}_{\varepsilon} = x$.)

Define an $\mathbb{A}_\nu$-valuation $\varepsilon_0 \in Valu_{\mathbb{A}_\nu}(M)$ by

$$\varepsilon_0(a) = 0^{\mathcal{M}} \text{ always.}$$

If t is a term, we write $\llbracket t\rrbracket^{\mathcal{M}}$ for the function $\lambda\varepsilon.\llbracket t\rrbracket^{\mathcal{M}}_{\varepsilon}$. If ξ is a sentence, we write $\llbracket \xi\rrbracket^{\mathcal{M}}$ for the function $\lambda\varepsilon.\llbracket \xi\rrbracket^{\mathcal{M}}_{\varepsilon}$.

We now define an interpretation $\mathcal{N}$ for $\dot{\mathcal{L}}$. We give a denotation to the base sorts ι and o of $\dot{\mathcal{L}}$, as follows:

$$\begin{aligned}
\iota^{\mathcal{N}} &= \{\llbracket t\rrbracket^{\mathcal{M}} \mid t \text{ a term of } \mathcal{L}{+}M\}\\
o^{\mathcal{N}} &= \{\llbracket \xi\rrbracket^{\mathcal{M}} \mid \xi \text{ a sentence of } \mathcal{L}{+}M\}
\end{aligned}$$

We give a denotation to the term-formers and proposition-formers of $\dot{\mathcal{L}}$, as follows:

$$\begin{aligned}
\mathsf{var}^{\mathcal{N}}\, a\, \varepsilon &= \varepsilon(a)\\
0^{\mathcal{N}}\, \varepsilon &= 0^{\mathcal{M}}\\
\mathsf{succ}^{\mathcal{N}}\, u\, \varepsilon &= succ^{\mathcal{M}}(u\varepsilon)\\
+^{\mathcal{N}}\, (u, v)\, \varepsilon &= +^{\mathcal{M}}(u\varepsilon, v\varepsilon)\\
*^{\mathcal{N}}\, (u, v)\, \varepsilon &= *^{\mathcal{M}}(u\varepsilon, v\varepsilon)\\
\mathsf{sub}^{\mathcal{N}}_{\iota}\, ([a]u, v)\, \varepsilon &= u(\varepsilon[a{::=}v\varepsilon])\\
\dot{\bot}^{\mathcal{N}}\, \varepsilon &= 0\\
\mathsf{sub}^{\mathcal{N}}_{o}\, ([a]u, v)\, \varepsilon &= U(\varepsilon[a{::=}v\varepsilon])\\
\dot{\Rightarrow}^{\mathcal{N}}\, (U, V)\, \varepsilon &= max\{1{-}U(\varepsilon), V(\varepsilon)\}\\
\dot{\forall}^{\mathcal{N}}\, [a]U\, \varepsilon &= min\{U(\varepsilon[a{::=}x]) \mid x \in M\}\\
\dot{\approx}^{\mathcal{N}}\, (u, v)\, \varepsilon &= \approx^{\mathcal{M}}(u\varepsilon, v\varepsilon)\\
\approx^{\mathcal{N}}_{\iota}\, (u, v) &= 1 \text{ if } u{=}v \text{ and } 0 \text{ otherwise}\\
\approx^{\mathcal{N}}_{o}\, (U, V) &= 1 \text{ if } U{=}V \text{ and } 0 \text{ otherwise}\\
\epsilon^{\mathcal{N}}\, U &= U(\varepsilon_0)
\end{aligned}$$

Here, u and v range over $\iota^{\mathcal{N}}$ and U and V range over $o^{\mathcal{N}}$.

Lemma 6.15. *1.* $\llbracket t'[a{::=}t]\rrbracket^{\mathcal{M}}_{\varepsilon} = \llbracket t'\rrbracket^{\mathcal{M}}_{\varepsilon[a::=\llbracket t\rrbracket^{\mathcal{M}}_{\varepsilon}]}$.
2. $\llbracket \xi[a{::=}t]\rrbracket^{\mathcal{M}}_{\varepsilon} = 1$ *if only if* $\llbracket \xi\rrbracket^{\mathcal{M}}_{\varepsilon[a::=\llbracket t\rrbracket^{\mathcal{M}}_{\varepsilon}]} = 1$.

Lemma 6.16. *The following equalities all hold:*

$$\begin{aligned}
\mathsf{var}^{\mathcal{N}}(a) &= \llbracket a\rrbracket^{\mathcal{M}}\\
0^{\mathcal{N}} &= \llbracket 0\rrbracket^{\mathcal{M}}\\
\mathsf{succ}^{\mathcal{N}}(\llbracket t\rrbracket^{\mathcal{M}}) &= \llbracket succ(t)\rrbracket^{\mathcal{M}}\\
+^{\mathcal{N}}(\llbracket t'\rrbracket^{\mathcal{M}}, \llbracket t\rrbracket^{\mathcal{M}}) &= \llbracket t' + t\rrbracket^{\mathcal{M}}\\
*^{\mathcal{N}}(\llbracket t'\rrbracket^{\mathcal{M}}, \llbracket t\rrbracket^{\mathcal{M}}) &= \llbracket t' * t\rrbracket^{\mathcal{M}}\\
\mathsf{sub}^{\mathcal{N}}_{\iota}([a]\llbracket t'\rrbracket^{\mathcal{M}}, \llbracket t\rrbracket^{\mathcal{M}}) &= \llbracket t'[a{::=}t]\rrbracket^{\mathcal{M}}\\
\mathsf{sub}^{\mathcal{N}}_{o}([a]\llbracket \xi\rrbracket^{\mathcal{M}}, \llbracket s\rrbracket^{\mathcal{M}}) &= \llbracket \xi[a{::=}s]\rrbracket^{\mathcal{M}}\\
\dot{\bot}^{\mathcal{N}} &= \llbracket \bot\rrbracket^{\mathcal{M}}\\
\dot{\Rightarrow}^{\mathcal{N}}(\llbracket \xi'\rrbracket^{\mathcal{M}}, \llbracket \xi\rrbracket^{\mathcal{M}}) &= \llbracket \xi' \Rightarrow \xi\rrbracket^{\mathcal{M}}\\
\dot{\forall}^{\mathcal{N}}([a]\llbracket \xi\rrbracket^{\mathcal{M}}) &= \llbracket \forall a.\xi\rrbracket^{\mathcal{M}}\\
\dot{\approx}^{\mathcal{N}}(\llbracket r\rrbracket^{\mathcal{M}}, \llbracket s\rrbracket^{\mathcal{M}}) &= \llbracket r \approx s\rrbracket^{\mathcal{M}}
\end{aligned}$$

Proof. We compare Definitions 6.14 and 6.10. Most cases are immediate; we consider only the slightly less trivial ones:

$$\begin{aligned}
\mathsf{var}^{\mathcal{N}}(a) &= (\lambda a.\lambda\varepsilon.\varepsilon(a))a && \text{Definition 6.14}\\
&= (\lambda a.\llbracket a\rrbracket^{\mathcal{M}})a && \text{Definition 6.10}\\
&= \llbracket a\rrbracket^{\mathcal{M}} && \text{fact}
\end{aligned}$$

$$\begin{aligned}
\mathsf{sub}^{\mathcal{N}}_{\iota}([a]\llbracket t'\rrbracket^{\mathcal{M}}, \llbracket t\rrbracket^{\mathcal{M}}) &= \lambda\varepsilon.\llbracket t'\rrbracket^{\mathcal{M}}(\varepsilon[a{::=}\llbracket t\rrbracket^{\mathcal{M}}\varepsilon]) && \text{Definition 6.14}\\
&= \lambda\varepsilon.\llbracket t'[a{::=}t]\rrbracket^{\mathcal{M}} && \text{Lemma 6.15}
\end{aligned}$$

Other cases are no harder. $\square$

Lemma 6.17. *$\mathcal{N}$ (Definition 6.14) is a PNL interpretation.*

Proof. We must check that:

$\iota^{\mathcal{N}}$ and $o^{\mathcal{N}}$ are permissive-nominal sets.

By routine calculations. (In fact, $\iota^{\mathcal{N}}$ and $o^{\mathcal{N}}$ are *nominal* sets; that is, their elements all have finite support.)

The functions defined in Definition 6.14 map elements of $\iota^{\mathcal{N}}$, $o^{\mathcal{N}}$, $[\mathbb{A}]\iota^{\mathcal{N}}$, and $[\mathbb{A}]o^{\mathcal{N}}$ correctly to the appropriate sets.

By Lemma 6.16.

$\epsilon^{\mathcal{N}}$ is equivariant from $o^{\mathcal{N}}$ to $\{0,1\}$.

By routine calculations using the fact that $(a\ b)\cdot\varepsilon_0 = \varepsilon_0$. $\square$

Lemma 6.18. *If $(\Xi \vdash \chi)^{\boldsymbol{\cdot}}$ is valid in $\mathcal{N}$, then $\Xi \vdash \chi$ is valid in $\mathcal{M}$.*

Proof. We calculate that if $(\Xi \vdash \chi)^{\boldsymbol{\cdot}}$ is valid in $\mathcal{N}$, then

$$[\![(\xi_1 \wedge \ldots \wedge \xi_k) \Rightarrow (\chi_1 \vee \ldots \vee \chi_l)]\!]^{\mathcal{M}}_{\varepsilon_0} = 1$$

But the proposition written out above is closed, so for all valuations ε, $[\![(\xi_1 \wedge \ldots \wedge \xi_k) \Rightarrow (\chi_1 \vee \ldots \vee \chi_l)]\!]^{\mathcal{M}}_{\varepsilon} = 1.$ $\square$

Proposition 6.19. *$\mathcal{N}$ is a model of $\mathcal{S} \cup \mathsf{ARITH}$.*

Proof. By a routine verification. We consider the axiom $(\dot\forall)$ from Figure 5. We unpack definitions and see that we must prove that for every ξ in $\mathcal{L}{+}M$, $\forall x \in M.\varepsilon_0[a::=x] \in (\xi)^{\boldsymbol{\cdot}}$ if and only if $\varepsilon_0[a::=(t)^{\boldsymbol{\cdot}}] \in (\xi)^{\boldsymbol{\cdot}}$ for every t a term of $\mathcal{L}{+}M$. This follows, because $\mathcal{L}{+}M$ has a constant symbol for every $x \in M$. Validity of the other axioms is no harder. $\square$

Theorem 6.20. $\mathsf{arithmetic}, \Xi \vdash \chi$ *in first-order logic if and only if $\mathcal{S} \cup \mathsf{ARITH} \vdash (\Xi \vdash \chi)^{\boldsymbol{\cdot}}$ in PNL.*

Proof. We prove two implications. The top-to-bottom implication follows using Theorem 6.6.

For the bottom-to-top implication, we reason as follows: Suppose $\mathcal{S} \cup \mathsf{ARITH} \vdash (\Xi \vdash \chi)^{\boldsymbol{\cdot}}$ in PNL. Choose an arbitrary interpretation $\mathcal{M}$ of first-order logic that is a model of arithmetic, with carrier set M. By Soundness (Theorem 5.15) and Proposition 6.19, $\vDash^{\mathcal{N}} (\Xi \vdash \chi)^{\boldsymbol{\cdot}}$. By Lemma 6.18 $\Xi \vdash \chi$ is valid in $\mathcal{M}$. $\mathcal{M}$ was arbitrary, so by completeness of first-order logic [29, §4.2] it follows that $\Xi \vdash \chi$ is derivable. $\square$

7. Other theories in PNL

We briefly sketch three other smaller, but relevant and interesting, PNL theories.

7.1 Inductive types

Permissive-nominal logic can express the principles of nominal abstract syntax developed in [22].

Suppose a base sort ι, a name sort ν, and term-formers $\mathsf{var} : \nu \to \iota$, $\mathsf{app} : (\iota, \iota) \to \iota$ and $\mathsf{lam} : [\nu]\iota \to \iota$. Fix an unknown $U : \iota$ and for brevity write $\phi[U::=r]$ as $\phi(r)$ for every ϕ. Suppose an axiom-scheme, for every ϕ:

$$\begin{array}{l}\phi(\mathsf{var}(a)) \Rightarrow \\ \forall X.(\phi(X) \Rightarrow \phi(\mathsf{lam}([a]X))) \Rightarrow \\ \forall X,Y.(\phi(X) \Rightarrow \phi(Y) \Rightarrow \mathsf{app}(X,Y)) \Rightarrow \\ \quad \forall X.(\phi(X))\end{array}$$

Here X and Y have sort ι and we make a fixed but arbitrary choice of atom $a \in p(X)$.

We can also express this finitely, if we axiomatise a sort for propositions (as we did for arithmetic). Here is the axiom-scheme above made finite by using the theories EQU, SUB, and FOL from Section 3:

$$\begin{array}{l}\forall Z.\epsilon(Z[a{\mapsto}\mathsf{var}(a)]) \Rightarrow \\ \quad \forall X.(\epsilon(Z[a{\mapsto}X]) \Rightarrow \epsilon(Z[a{\mapsto}\mathsf{lam}([a]X))) \Rightarrow \\ \quad \forall X,Y.(\epsilon(Z[a{\mapsto}X]) \Rightarrow \epsilon(Z[a{\mapsto}Y]) \Rightarrow \epsilon(Z[a{\mapsto}\mathsf{app}(X,Y)])) \Rightarrow \\ \quad\quad \forall X.\epsilon(Z[a{\mapsto}X])\end{array}$$

7.2 Freshness: from $a \notin fa(r)$ to $a \notin supp(x)$

PNL has a notion of 'free atoms of' $a \in fa(\phi)$. Nominal sets have rather similar-looking notion of support $a \in supp(x)$.

Support, or rather its negation *freshness*, is definable using equality, intuitively as

> "a is fresh for x when $(b\ a)\cdot x = x$ for some/any fresh b."

See e.g. equation 13 in [22].

To capture a freshness predicate for a name sort ν on a sort α in PNL, we assume EQU (or extend PNL with an equality primitive) and assume a predicate $\#$ of arity (ν, α) with an axiom

$$\forall X.(a\#X \Leftrightarrow (b\,a)\cdot X \approx X).$$

Here X has sort α and a and b have sort ν, and $a \in p(X)$ and $b \notin p(X)$.

A detailed discussion of support as a definitional extension of equational reasoning is in Sections 5.2 and 6.1.1 of [18].

The reader should distinguish the notion 'free atoms' from the nominal sets notion 'support'. These are two distinct concepts and they are not the same.

7.3 The rule (И), and the И quantifier

Nominal sets support the И-quantifier [22]. PNL does too, but in an unexpected and perhaps rather elegant manner.

И has some distinctive properties which are reflected in nominal logic (NL) and the logic of FM sets (FM):

$$\frac{\forall x.(\mathsf{P}(x) \Rightarrow \text{И}a.\mathsf{Q}(a,x))}{\forall x.\text{И}a.(\mathsf{P}(x) \Rightarrow \mathsf{Q}(a,x))} \qquad \frac{\forall x.\text{И}a.\text{И}b.(b\,a)\cdot x{\approx}x}{\text{И}a.\text{И}b.\forall x.(a\#x \Rightarrow b\#x \Rightarrow (b\,a)\cdot x{\approx}x)}$$

Here and below we write a double horizontal line for 'is provably equivalent to'. In PNL И is 'hiding' in the permission sets. Corresponding propositions are, where $a, b \notin p(X)$:

$$\frac{\forall X.(\mathsf{P}(X) \Rightarrow \mathsf{Q}(a,X))}{\forall X.(\mathsf{P}(X) \Rightarrow \mathsf{Q}(a,X))} \qquad \frac{\forall X.(b\,a)\cdot X \approx X}{\forall X.(b\,a)\cdot X \approx X}$$

Two things are happening: freshness conditions are hard-coded into the syntax by permission sets — and so is И.

It is interesting to consider another example. In NL/FM:

$$\frac{\text{И}a.\mathsf{P}(a) \wedge \text{И}a.\mathsf{Q}(b)}{\text{И}a.\text{И}b.(\mathsf{P}(a) \wedge \mathsf{Q}(b))} \qquad \frac{\text{И}a.\mathsf{P}(a) \wedge \text{И}a.\mathsf{Q}(b)}{\text{И}a.(\mathsf{P}(a) \wedge \mathsf{Q}(a))}$$

Correspondingly in PNL:

$$\frac{\mathsf{P}(a) \wedge \mathsf{Q}(b)}{\mathsf{P}(a) \wedge \mathsf{Q}(b)} \qquad \frac{\mathsf{P}(a) \wedge \mathsf{Q}(b)}{\mathsf{P}(a) \wedge \mathsf{Q}(a)}$$

It is easy to use the rule (И) from Figure 2 to construct a derivation proving that $\mathsf{P}(a) \wedge \mathsf{Q}(b)$ and $\mathsf{P}(a) \wedge \mathsf{Q}(a)$ are indeed logically equivalent in PNL.

(И) expresses that truth is preserved by permutative renaming, or in symbols: $\vdash \phi \Leftrightarrow \pi\cdot\phi$ always. We call this rule

(И) because of another way of looking at matters: every atom in ϕ is abstracted by a generalised И-abstraction. Permission sets, or perhaps rather their complements, are a freshness quantifier.

8. Related work

Nominal terms Nominal terms were introduced in [31] where a decidable and efficient unification algorithm was demonstrated (see [3] for the state of the art). Nominal terms were then used in specification languages; in rewriting [10], and in universal algebra (the logic of equality) [18].

Nominal terms require a finite *freshness context* $a\#X$. These are explicit assertions corresponding to $a \notin p(X)$.

Contrast a prototype of permissive-nominal term unification [25] with the Haskell Nominal Toolkit (HNT) [2]. Core functions in the PNL implementation (e.g. unification and alpha-equivalence checks) are pure. The types of their HNT counterparts are monadic because of explicit freshness contexts. The freshness context becomes a notion of state, which must be passed around in a monad.

Compare the types (1) of the alpha-equivalence check function from the HNT and (2) of the implementation of permissive-nominal terms:[2]

$$\begin{array}{lll}(1) & \texttt{alpha'check} :: & (\texttt{Show}\ t,\ \texttt{Eq}\ t,\ \texttt{Ord}\ a,\ \texttt{Ord}\ v) \Rightarrow \texttt{FrsCtxt}\ v\ a \to \\ & & \texttt{Term}\ a\ t\ v \to \texttt{Term}\ a\ t\ v \to \\ & & \texttt{CS}\ r\ (\texttt{ExtB}\ l\ e\ (\texttt{ErrorT}\ [\texttt{Char}]))\ m\ () \\ (2) & \texttt{aeq} \qquad\quad :: & (\texttt{Eq}\ a, \texttt{Permissive}\ b) \Rightarrow \texttt{Term}\ a\ b \to \texttt{Term}\ a\ b\ \to \texttt{Bool}\end{array}$$

The type associated with the 'permissive' version is shorter. To be fair, HNT does a lot more and this is also responsible for some complexity. Nevertheless, we have seen a similar phenomenon, which has been truly inconvenient, in our own work [10, 11, 15, 17, 20].

Relative to nominal terms, PNL does without freshness contexts and adds quantification over unknowns.

Permissive-nominal terms Permissive-nominal terms were developed to address the concerns above [6, 7]. They simplify α-equality and freshness and separate them from equational reasoning. There is no 'freshness monad'.

Permissive-nominal terms correspond to the term-language of PNL. Their unification algorithm, which is decidable and closer to the first-order case than that of nominal terms, extends easily to the syntax of PNL.

Relative to [7], we add first-order logic and in particular universal quantification $\forall X$ over unknowns.

Nominal logic The two best-known 'nominal' logics are probably the *nominal logic* of [28] and *FM set theory*. Both are Hilbert-style theories — sets of axioms — in first-order logic.

FM set theory contains axioms whose intended model is the cumulative hierarchy of hereditarily finitely supported sets. Nominal logic contains axioms whose intended model is sets with a finitely-supported permutation action.

Natural deduction rules for И were proposed e.g. in [22, Proposition 4.10], but these are not closed under substitution. The second author created a proof-theory for И [12] with a good notion of proof-normalisation and a completeness proof, followed by an alternative treatment with Cheney [14]. These did not give И a direct operational behaviour as 'pick a globally fresh name'; this was captured by Cheney in a later paper [4].

Cheney's logic involves 12 infinite axiom-schemes (see Figures 3 and 4 of [4]). These describe atoms-abstraction as an equality theory.

PNL handles α-equivalence, and the И-quantifier, without recourse to axioms, and indeed, without requiring an equality or a И-quantifier. We can 'just α-rename'; freshness and renaming are separated from equality reasoning in the logic.

Nominal logic from [28] is not complete for its semantics in nominal sets, for reasons discussed and addressed by Cheney in [5]. In fact, PNL is complete for the semantics in this paper; permissive-nominal sets with their infinite support are clearly related to Cheney's *support ideals*.

The PNL proof system is 'ϵ away' from that of first-order logic. Arguably, it is more immediate to first-order logic than any other previous reasoning system based on nominal sets.

One-and-a-halfth order logic from [17] This logic is designed to represent schematic first-order reasoning (first-order derivations in the presence of 'unknown predicates'). It corresponds roughly to the axiomatisation of first-order logic in Section 3.

Semantic nominal terms In [19] we show how to interpret level 2 variables (unknowns) as infinite lists of distinct level 1 variables (atoms). This allows us to build permissive-nominal term syntax as nominal abstract syntax-style inductive datatypes as proposed in [22]. The aim of this paper is to discuss the logic; not to analyse how its syntax could best be built.

[19] does a few other things, and one of them is relevant to the logic of PNL: it defines a capturing and capture-avoiding substitution, which we plan to import into PNL in a later journal version of this work.

First- and higher-order logic As discussed in the Introduction, we see PNL as sitting between these two logics. It is better than first-order logic, in that term-formers can bind. It is better than higher-order logic in that its models are smaller and simpler, and its syntax supports a decidable unification algorithm.

PNL syntax and derivations can be translated to higher-order logic as nominal algebra can [21], though syntax may undergo a quadratic increase in size in the translation. We map to a *pattern fragment* of full higher-order syntax.

The issue is how naturally the structure of our intended axioms fits the structure of the logic — and also how naturally the semantics reflects any intuitions we have about those axioms. We feel that PNL does a very nice job of being first order and making binding convenient to work with in syntax and semantics.

Logics based on the ∇-quantifier A family of logics based on the ∇-quantifier [23, 24, 30] has been developed specifically for reasoning on inductive datatypes with binding. Permissive-nominal logic can do this too, and because of its 'nominal' ancestry it does so in the style of nominal techniques.

Logics based on ∇ use *raising* to obtain the effect of capturing substitution and variable dependencies, whereas we use a two-level term syntax.

∇ commutes with $\forall$-quantification and the И-quantifier does not; PNL does away with the И-quantifier altogether.

An interesting *similarity* is that some recent work in this field has also moved to using a syntax with name-constants

[2] See the documentation for module `Nominal.Matching` at `http://www.dcs.kcl.ac.uk/pg/calves/hnt/doc/Nominal-Matching.html` and module `Terms.Terms` in the permissive-nominal terms implementation source code, available at `http://www.macs.hw.ac.uk/~dpm8/permissive/`

and equivariance, reminiscent of the atoms and rule ($\reflectbox{N}$) [30]. We would like to give these logics models in nominal sets, which would give us a better view of where they stand in relation to our own work.

Contextual modal type theory (CMTT) Logics based on CMTT are consistent and have a well-studied proof-theory, so models can be constructed using normal forms. We do not know of what general class of possibly non-syntactic structures these syntactic models are instances. Developing such models would be interesting future work.

CMTT is a two-level system (contexts split into two halves; Δ and Φ) but the two levels do a different job from the two levels used in (permissive-)nominal terms. Variables $u : A[\Phi] \in \Delta$ range over representations of code; $x : A \in \Phi$ range over denotation.

To the extent that this could be represented in PNL at all, it would be represented in the sorts — one sort for code, another for denotation (values).

Further logics Coming from other threads of research in computer science, there have been logics designed to enrich first-order logic directly with binders. We note *binding logic* [8] and λ-*logic* [1].

Binding logic enriched first-order terms with binders but forbade capture and turned out to be a little too weak.

λ-logic takes a direct approach of enriching first-order terms with λ-abstraction. The approach to binding taken by PNL is somewhat more general and is certainly different, in that it allows us to treat names as 'bindable data'. That is, we can compare names for *in*equality as names, while at the same time we can give them the behaviour of variables by axiomatising e.g. substitution for them, if we wish.

9. Further remarks, further work

Unknowns of name sort, and atoms Should every inhabitant of the semantics of a name sort ν be referenced by an atom? Should every inhabitant of the semantics of name sorts be referenced by a closed term? In PNL as it stands, neither is necessarily the case.

This is only as much of an issue as we choose to make of it. It is not unusual for there to be more elements in a type than there are closed terms. It is a fact that PNL is complete for the semantics of this paper, so extra elements in name sorts cannot 'make anything false'.

We can view 'non-standard atoms' like as like 'non-standard numbers' in first-order arithmetic; a price we pay for the first-order aspect of the language.

It is still reasonable to ask whether we can exclude these 'non-standard atoms'. In fact, we believe that this is possible. The idea is in the proof-theory for $\reflectbox{N}$ from 'Fresh Logic' [12]; see (**Exhaust**$\mathbb{A}$) in Figure 3, and Subsection 5.5. This is a topic for future research.

Extending sorts It would be a good idea to introduce sort-formers and polymorphism into the sorting system, so that e.g. we can conveniently axiomatise a substitution action on an infinite class of sorts. We see no difficulty in doing this — it is a definitional extension of what we already have.

We might also assume, for every sort α, an associated name sort ν_α. In particular, this would allow us to conveniently talk about 'atoms (level 1 variables) associated with α' in the same way that we can talk about 'level 2 variables of sort α'.

Size of permission sets There is design freedom in the choice of permission sets. For example, we can take $\mathbb{A}$ uncountable and permission sets all countably infinite sets, or we can take permission sets $\{\pi(a) \mid a \in \mathbb{A}^{<} \setminus A,\ A \subseteq \mathbb{A} \text{ finite, all } \pi\}$. We do not currently see a 'right' choice. The choice in Definition 2.4 is simple.

Note the following: Suppose a base sort τ, a name sort ν and $a : \nu$, a proposition-former $\mathsf{P} : \tau$, and a function-symbol $\mathsf{f} : (\tau, \nu)\tau$. In spite of this, $\forall X.\mathsf{P}(X) \not\vdash \mathsf{P}(\mathsf{f}(X, a))$ where $a \notin p(X)$, since no finite permutation maps $p(X)$ to $fa(\mathsf{f}(X, a)) = p(X) \cup \{a\}$.

On closed terms (with no unknowns), this is a non-issue because the syntax is finite and mentions finitely many atoms. Closed terms will be sufficient for many purposes; for example closed terms contains the image of the translation of first-order arithmetic in this paper; any sensible choices of permission set will thus allow us to write axioms as we have done in this paper.

In [19] we generalise the substitution action $[X{::=}r]$ to the case that $fa(r) \subseteq p(X)$ is not necessarily true. Developing this is future work.

Summary Permissive-nominal logic comes out of research into inductive definitions with names and binding in mechanised mathematics. It can also be applied to these problems.

But PNL provides a logical environment which, we feel, does a very good job of modelling the 'informal meta-level', complete with names, binding, and first-order reasoning.

Arguably its most exciting potential application is as a logical foundation — as a meta-theory for mathematics — intermediate in power between first- and higher-order logic. The case studies in this paper suggest that this could be feasible. We believe that, perhaps with some fairly modest extensions, it would make an expressive, ergonomic, and practical alternative meta-language for mechanised mathematics.

References

[1] Michael Beeson. Lambda logic. In *Second International Joint Conference on Automated Reasoning (IJCAR 2004)*, volume 3097 of *Lecture Notes in Computer Science*, pages 460–474. Springer, 2004.

[2] Christophe Calvès. A Haskell nominal toolkit. In *Proceedings of the 2nd International Workshop on Theory and Applications of Abstraction, Substitution and Naming (TAASN 2009)*, 2009.

[3] Christophe Calvès and Maribel Fernández. A polynomial nominal unification algorithm. *Theoretical Computer Science*, 403:285–306, August 2008.

[4] James Cheney. A simpler proof theory for nominal logic. In *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 2005.

[5] James Cheney. Completeness and Herbrand theorems for nominal logic. *Journal of Symbolic Logic*, 71:299–320, 2006.

[6] Gilles Dowek, Murdoch J. Gabbay, and Dominic P. Mulligan. Permissive Nominal Terms and their Unification. In *Proceedings of the 24th Italian Conference on Computational Logic (CILC'09)*, 2009.

[7] Gilles Dowek, Murdoch J. Gabbay, and Dominic P. Mulligan. Permissive Nominal Terms and their Unification (journal version). *Logic Journal of the IGPL*, 2010. In press.

[8] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Binding logic: Proofs and models. In *Proceedings of the 9th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2002)*, pages 130–144. Springer, 2002.

[9] William M. Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 3(6):267–286, 2008.

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

[11] Murdoch J. Gabbay. A NEW calculus of contexts. In *Proceedings of the 7th ACM SIGPLAN symposium on Principles and Practice of Declarative Programming (PPDP 2005)*, pages 94–105. ACM, 2005.

[12] Murdoch J. Gabbay. Fresh Logic. *Journal of Applied Logic*, 5(2):356–387, June 2007.

[13] Murdoch J. Gabbay. A General Mathematics of Names. *Information and Computation*, 205(7):982–1011, July 2007.

[14] Murdoch J. Gabbay and James Cheney. A Sequent Calculus for Nominal Logic. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 139–148. IEEE Computer Society, 2004.

[15] Murdoch J. Gabbay and Stéphane Lengrand. The lambda-context calculus (extended version). *Information and computation*, 207:1369–1400, 2009.

[16] Murdoch J. Gabbay and Aad Mathijssen. Capture-Avoiding Substitution as a Nominal Algebra. *Formal Aspects of Computing*, 20(4-5):451–479, June 2008.

[17] Murdoch J. Gabbay and Aad Mathijssen. One-and-a-halfth-order Logic. *Journal of Logic and Computation*, 18(4):521–562, August 2008.

[18] Murdoch J. Gabbay and Aad Mathijssen. Nominal universal algebra: equational logic with names and binding. *Journal of Logic and Computation*, 19(6):1455–1508, December 2009.

[19] Murdoch J. Gabbay and Dominic P. Mulligan. Semantic nominal terms. In *TAASN*, 2009.

[20] Murdoch J. Gabbay and Dominic P. Mulligan. Two-level lambda-calculus. In *Proceedings of the 17th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2008)*, volume 246, pages 107–129, August 2009.

[21] Murdoch J. Gabbay and Dominic P. Mulligan. Universal algebra over lambda-terms and nominal terms: the connection in logic between nominal techniques and higher-order variables. In *Proceedings of the 4th International Workshop on Logical Frameworks and Meta-Languages (LFMTP 2009)*, pages 64–73. ACM, 2009.

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

[23] Andrew Gacek, Dale Miller, and Gopalan Nadathur. Combining generic judgments with recursive definitions. In *Proceedings of the 23rd IEEE Symposium on Logic in Computer Science (LICS 2008)*, pages 33–44. IEEE Computer Society Press, 2008.

[24] Dale Miller and Alwen Tiu. A proof theory for generic judgments (extended abstract). In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 118–127. IEEE Computer Society Press, 2003.

[25] Dominic P. Mulligan. Implementation of permissive nominal terms. Available at `http://www2.macs.hw.ac.uk/~dpm8/permissive/perm.htm`, 2009.

[26] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.

[27] Lawrence C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[28] Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186(2):165–193, 2003.

[29] Joseph Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.

[30] Alwen Tiu. A logic for reasoning about generic judgments. *Electronic Notes in Theoretical Computer Science*, 174(5):3–18, 2007.

[31] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal Unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.