The Semantics of Nominal Logic Programs

James Cheney

University of Edinburgh
jcheney@inf.ed.ac.uk

Abstract. Nominal logic programming is a form of logic programming with "concrete" names and binding, based on nominal logic, a theory of alpha-equivalence founded on swapping and freshness constraints. Previous papers have employed different characterizations of the semantics of nominal logic programs, including operational, denotational, and proof-theoretic characterizations. In this paper we give a uniform presentation of these characterizations and prove appropriate soundness and completeness results. We also give some applications of these results.

1 Introduction

Nominal logic is an extension of first-order logic that provides support for programming with abstract syntax with names and binding modulo α-equivalence. It is similar in spirit to higher-order logic programming languages and logical frameworks that provide *higher-order abstract syntax* facilities for encoding variables and binding in object languages; however, nominal logic is semantically much closer to first-order logic. Nominal logic axiomatizes α-equivalence in terms of an invertible renaming operation ("name-swapping") and the freshness relation; it also includes a novel self-dual "fresh name" quantifier $\reflectbox{$\mathsf{N}$}$.

In previous work, Cheney and Urban [1] have presented αProlog, a logic programming language inspired by nominal logic, which employs a simple backchaining proof search technique like that of Prolog, but uses Urban, Pitts, and Gabbay's *nominal unification* [16] algorithm instead of first-order (that is, purely syntactic) unification. In addition, αProlog permits the $\reflectbox{$\mathsf{N}$}$-quantifier in both goal formulas and clauses. The $\reflectbox{$\mathsf{N}$}$-quantifier can be used to write some programs more easily than in higher-order abstract syntax, specifically programs that involve inequalities among names (for example, closure conversion).

This paper is concerned with establishing the relationships among operational, proof-theoretic, and denotational semantics of nominal logic programs. Because nominal logic involves both equality and freshness constraints, the appropriate semantic framework is that of constraint logic programming. Nominal logic can be presented as a theory of first-order logic [15]; indeed, the $\reflectbox{$\mathsf{N}$}$-quantifier can be defined in terms of the existential (or equivalently universal) quantifiers and the freshness relation (#):

$$\reflectbox{N}\mathsf{a}.\phi(\mathsf{a},\overline{x}) \iff \exists a.a \mathrel{\#} \overline{x} \wedge \phi(a,\overline{x}) \iff \forall a.a \mathrel{\#} x \supset \phi(a,\overline{x})$$

Moreover, as shown in a separate paper, nominal logic has a well-behaved Herbrand model theory and nominal terms form a well-behaved constraint domain. Consequently, we can view Horn clause nominal logic programming as an instance of the Constraint Logic Programming Scheme [9], by translating И-quantified goal subformulas to equivalent existential formulas and И-quantified program clauses to universal formulas.

While this approach suffices to define a semantics for nominal logic programs, it is unsatisfactory in some respects. First, the syntactic translation that removes И may increase the size of the program by a quadratic factor, thus slowing down analyses formulated in terms of the CLP semantics. Second, from prior work on the proof theory of nominal logic [8, 3], we know that both И-goals and program clauses can be read as a proof search operation "generate a fresh name and proceed to solve goal G (or refine clause D)". However, the translation of И obscures this proof-search reading. Finally, nominal constraint solving is NP-complete [2], whereas Urban, Pitts, and Gabbay's nominal unification algorithm solves a polynomial time special case. However their algorithm relies on the use of "atoms" (special name-constants which correspond to И-quantified names in nominal logic programs). Translating atoms to ordinary first-order variables precludes the use of this efficient constraint solving algorithm. We believe that these facts justify the investigation of more direct approaches to the semantics of nominal logic programs.

Several such direct semantics have been presented in previous papers on nominal logic and αProlog, including an operational semantics [1], a denotational semantics [5, Chapter 6] and two proof-theoretic semantics [8, 17]. In [5], soundness, algebraic completeness, and logical completeness results for the denotational and operational semantics paralleling Jaffar et al's semantics of constraint logic programming [9] were developed. However, except for this work, the relationships among the approaches have not been studied carefully.

The purpose of this paper is to fill this gap, by giving a uniform presentation of the denotational, proof-theoretic, and operational semantics and proving appropriate soundness and completeness theorems among them. Section 2 reviews nominal logic and nominal Horn clause programs. Section 3 presents the denotational semantics; Section 4 the proof-theoretic semantics; and Section 5 the operational semantics. Appropriate soundness and completeness theorems are proved along the way. Section 6 sketches two applications of these results, and Sections 7 and 8 discuss related and future work and conclude.

2 Background

The syntax of nominal logic is shown in Figure 1. We assume fixed countable sets of *variables* $\mathbb{V}$ and *names* $\mathbb{A}$. A *language* $\mathcal{L}$ consists of a set of *data types* δ, *name types* ν, *constants* $c : \delta$, *function symbols* $f : \overline{\sigma} \to \delta$, and *relation symbols* $p : \overline{\sigma} \to o$. The novel term constructors include names $\mathsf{a} \in \mathbb{A}$, name-abstractions $\langle a \rangle t$ denoting α-equivalence classes, and name-swapping applications $(a\ b) \cdot t$. The atomic formula $a \# t$ is called *freshness*, and intuitively means that the name

$$
\begin{array}{ll}
\text{(Types)} & \sigma ::= \nu \mid \delta \mid \langle\nu\rangle\sigma \\
\text{(Contexts)} & \Sigma ::= \cdot \mid \Sigma, x{:}\sigma \mid \Sigma\#\mathsf{a}{:}\nu \\
\text{(Terms)} & t ::= \mathsf{a} \mid c \mid f(\overline{t}) \mid x \mid (a\ b)\cdot t \mid \langle a\rangle t \\
\text{(Constraints)} & C ::= t \approx u \mid a \mathrel{\#} t \mid C \wedge C' \mid \exists X{:}\sigma.C \mid \text{И}\mathsf{a}{:}\sigma.C \\
\text{(Formulas)} & \phi ::= \top \mid \bot \mid p(\overline{t}) \mid C \mid \phi \supset \psi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \forall x{:}\sigma.\phi \mid \exists x{:}\sigma.\phi \mid \text{И}\mathsf{a}{:}\nu.\phi
\end{array}
$$

Fig. 1. Syntax of nominal logic

$$
\frac{\mathsf{a}:\nu \in \Sigma}{\Sigma \vdash \mathsf{a}:\nu} \quad
\frac{x:\sigma \in \Sigma}{\Sigma \vdash x:\sigma} \quad
\frac{c:\delta \in \mathcal{L}}{\Sigma \vdash c:\delta} \quad
\frac{f:\overline{\sigma} \to \delta \in \mathcal{L} \quad \Sigma \vdash \overline{t}:\overline{\sigma}}{\Sigma \vdash f(\overline{t}):\delta} \quad
\frac{\Sigma \vdash a:\nu \quad \Sigma \vdash t:\sigma}{\Sigma \vdash \langle a\rangle t:\langle\nu\rangle\sigma}
$$

$$
\frac{\Sigma \vdash a:\nu \quad \Sigma \vdash b:\nu \quad \Sigma \vdash t:\sigma}{\Sigma \vdash (a\ b)\cdot t:\sigma} \quad
\frac{\Sigma \vdash t,u:\sigma}{\Sigma \vdash t \approx u:o} \quad
\frac{\Sigma \vdash a:\nu \quad \Sigma \vdash t:\sigma}{\Sigma \vdash a \mathrel{\#} t:o}
$$

$$
\frac{}{\Sigma \vdash \top,\bot:o} \quad
\frac{\Sigma \vdash \phi,\psi:o}{\Sigma \vdash \phi\wedge\psi, \phi\vee\psi, \phi\supset\psi:o} \quad
\frac{\Sigma, X{:}\sigma \vdash \phi:o}{\Sigma \vdash \forall X{:}\sigma.\phi, \exists X{:}\sigma.\phi:o} \quad
\frac{\Sigma\#\mathsf{a}{:}\nu \vdash \phi:o}{\Sigma \vdash \text{И}\mathsf{a}{:}\nu.\phi:o}
$$

Fig. 2. Well-formedness for nominal terms and formulas

$$
\begin{array}{lll}
(\mathsf{a}\ \mathsf{b})\cdot\mathsf{a} = \mathsf{b} & & (\mathsf{a}\ \mathsf{b})\cdot c = c \\
(\mathsf{a}\ \mathsf{b})\cdot\mathsf{b} = \mathsf{a} & & (\mathsf{a}\ \mathsf{b})\cdot f(\overline{t}) = f((\mathsf{a}\ \mathsf{b})\cdot\overline{t}) \\
(\mathsf{a}\ \mathsf{b})\cdot\mathsf{a}' = \mathsf{a}' & (\mathsf{a} \neq \mathsf{a}' \neq \mathsf{b}) & (\mathsf{a}\ \mathsf{b})\cdot\langle\mathsf{a}'\rangle t = \langle(\mathsf{a}\ \mathsf{b})\cdot\mathsf{a}'\rangle(\mathsf{a}\ \mathsf{b})\cdot t
\end{array}
$$

$$
\frac{(\mathsf{a} \neq \mathsf{b})}{\vDash \mathsf{a} \mathrel{\#} \mathsf{b}} \quad
\frac{}{\vDash \mathsf{a} \mathrel{\#} c} \quad
\frac{\bigwedge_{i=1}^{n} \vDash \mathsf{a} \mathrel{\#} t_i}{\vDash \mathsf{a} \mathrel{\#} f(t_1^n)} \quad
\frac{\vDash \mathsf{a} \mathrel{\#} \mathsf{b} \quad \vDash \mathsf{a} \mathrel{\#} t}{\vDash \mathsf{a} \mathrel{\#} \langle\mathsf{b}\rangle t} \quad
\frac{}{\vDash \mathsf{a} \mathrel{\#} \langle\mathsf{a}\rangle t}
$$

$$
\frac{}{\vDash \mathsf{a} \approx \mathsf{a}} \quad
\frac{}{\vDash c \approx c} \quad
\frac{\bigwedge_{i=1}^{n} \vDash t_i \approx u_i}{\vDash f(t_1^n) \approx f(u_1^n)} \quad
\frac{\vDash t \approx u}{\vDash \langle\mathsf{a}\rangle t \approx \langle\mathsf{a}\rangle u} \quad
\frac{\vDash \mathsf{a} \mathrel{\#} u \quad \vDash t \approx (\mathsf{a}\ \mathsf{b})\cdot u}{\vDash \langle\mathsf{a}\rangle t \approx \langle\mathsf{b}\rangle u}
$$

Fig. 3. Swapping, freshness, and equality for ground nominal terms

a is not free in the term t. Well-formed terms and atomic formulas are defined in Figure 2. Contexts include ordinary typed variables $\Sigma, X{:}\sigma$ and name-typed names $\Sigma\#\mathsf{a}{:}\nu$. We write o for the type of propositions; quantification over types mentioning o is not allowed.

Figure 3 defines the swapping, freshness, and equality operations on ground terms. Swapping exchanges two syntactic occurrences of a name in a term (including occurrences such as a in $\langle\mathsf{a}\rangle t$.) The freshness relation defines what it means for a name to be "not free in" (or *fresh* for) a term. Intuitively, a name a is fresh for a term t (that is, $\mathsf{a} \mathrel{\#} t$) if t possesses no occurrences of a unenclosed by an abstraction of a. Finally, the equality relation on nominal terms is defined using freshness and swapping. The only interesting cases are for abstractions; the second rule for abstractions is equivalent to more standard forms of α-renaming, as has been shown elsewhere [15].

We sometimes refer to the set of "free" names of a term $supp(t) = \mathbb{A} - \{\mathsf{a} \mid \mathsf{a} \mathrel{\#} t\}$ as its *support*. Also, swapping and support are extended to formulas by setting $(\mathsf{a}\ \mathsf{b})\cdot Qx.\phi[x] = Qx.(\mathsf{a}\ \mathsf{b})\cdot\phi[x]$ for $q \in \{\forall,\exists\}$ and $(\mathsf{a}\ \mathsf{b})\cdot \text{И}\mathsf{a}'.\phi = \text{И}\mathsf{a}'.(\mathsf{a}\ \mathsf{b})\cdot\phi$, provided $\mathsf{a}' \notin \{\mathsf{a},\mathsf{b}\}$; e.g. $(\mathsf{a}\ \mathsf{b})\cdot\forall x.p(\mathsf{a},\mathsf{b},x) = \text{И}\mathsf{a}'.\forall x.p(\mathsf{a}',\mathsf{a},x)$.

$$
\begin{array}{lll@{\qquad}lll}
\mathcal{H} \vDash \top & & & \mathcal{H} \vDash \phi \vee \psi & \iff & \mathcal{H} \vDash \phi \text{ or } \mathcal{H} \vDash \psi \\
\mathcal{H} \nvDash \bot & & & \mathcal{H} \vDash \phi \supset \psi & \iff & \mathcal{H} \vDash \phi \text{ implies } \mathcal{H} \vDash \psi \\
\mathcal{H} \vDash A & \iff & A \in \mathcal{H} & \mathcal{H} \vDash \forall x{:}\sigma.\phi & \iff & \text{for all } t : \sigma,\ \mathcal{H} \vDash \phi[t/x] \\
\mathcal{H} \vDash t \approx u & \iff & \vDash t \approx u & \mathcal{H} \vDash \exists x{:}\sigma.\phi & \iff & \text{for some } t : \sigma,\ \mathcal{H} \vDash \phi[t/x] \\
\mathcal{H} \vDash a \mathrel{\#} u & \iff & \vDash a \mathrel{\#} u & \mathcal{H} \vDash \text{И}\mathsf{a}{:}\nu.\phi & \iff & \text{for fresh } \mathsf{b} : \nu \notin \mathit{supp}(\text{И}\mathsf{a}{:}\nu.\phi), \\
\mathcal{H} \vDash \phi \wedge \psi & \iff & \mathcal{H} \vDash \phi \text{ and } \mathcal{H} \vDash \psi & & & \mathcal{H} \vDash (\mathsf{b}\ \mathsf{a}) \cdot \phi.
\end{array}
$$

Fig. 4. Term model semantics of nominal logic

Likewise, swapping can be extended to sets of terms or formulas by setting $(\mathsf{a}\ \mathsf{b}) \cdot S = \{(\mathsf{a}\ \mathsf{b}) \cdot x \mid x \in S\}$.

For the purposes of this paper, it suffices to restrict attention to *term models* of nominal logic in which the domain elements are nominal terms with equality and freshness defined as in Figure 3. We write $B_{\mathcal{L}}$ for the *Herbrand base*, that is, the set of all non-constraint atomic formulas. We view an Herbrand model $\mathcal{H}$ as a subset of $B_{\mathcal{L}}$ that is *equivariant*, or closed under swapping (that is, $A \in \mathcal{H}$ implies $(\mathsf{a}\ \mathsf{b}) \cdot A \in \mathcal{H}$ for any a, b.) The semantics of nominal logic formulas over term models is defined as shown in Figure 4. The only nonstandard case is that for И; it is shown in [4] that this definition of the semantics of И-quantified formulas is correct for term models.

We generalize the satisfiability judgments as follows. Given sets of closed formulas Γ, Δ, we write $\mathcal{H} \vDash \Gamma$ to indicate that $\mathcal{H} \vDash \phi$ for each $\phi \in \Gamma$, and $\Gamma \vDash \Delta$ to indicate that $\mathcal{H} \vDash \Gamma$ implies $\mathcal{H} \vDash \Delta$. We define ground substitutions θ as functions from $\mathbb{V}$ to ground terms. Given a context Σ, we say that a ground substitution θ *satisfies* Σ when $\theta(x) : \Sigma(x)$ for all $x \in \Sigma$, and $\mathsf{a} \mathrel{\#} \sigma(x)$ for every subcontext $\Sigma' \# \mathsf{a}$ and every $x \in \Sigma'$. For example, $[X \mapsto \mathsf{a}, Y \mapsto \mathsf{b}]$ satisfies $\Sigma = \mathsf{a}, X \# \mathsf{b}, Y$ but not $X, Y \# \mathsf{a} \# \mathsf{b}$. We write $\Sigma : \theta \vDash \phi$, $\Gamma, \theta \vDash \Delta$, etc. as shorthand for $\theta : \Sigma$ and $\vDash \theta(\phi)$, $\theta(\Gamma) \vDash \theta(\Delta)$, etc. Moreover, we write $\Sigma : \Gamma \vDash \Delta$ to indicate that $\Sigma : \Gamma, \theta \vDash \Delta$ for every $\theta : \Sigma$. Note that, for example, $X \# \mathsf{a} : \cdot \vDash \mathsf{a} \mathrel{\#} X$ but $\mathsf{a}, X : \cdot \nvDash \mathsf{a} \mathrel{\#} X$.

We define the goal formulas G and program clauses D of *nominal logic programs* as follows:

$$
\begin{aligned}
G &::= \top \mid A \mid C \mid G \wedge G' \mid G \vee G' \mid \exists X.G \mid \text{И}\mathsf{a}.G \\
D &::= \top \mid A \mid D \wedge D' \mid G \supset D \mid \forall X.D \mid \text{И}\mathsf{a}.D
\end{aligned}
$$

We refer to these as *nominal Horn goals/program clauses*. For many examples of such programs, see [1] or [5, Chapter 2].

3 Denotational Semantics

In this section we define the denotational semantics of nominal logic programs. We now show that least Herbrand models exist for nominal Horn clause programs and that the least Herbrand model is the least fixed point of an appropriate one-step deduction operator, following Lloyd [11]. This section relies on standard definitions and concepts from lattice theory [7].

3.1 Least Herbrand Models

It is a well-known fact that least Herbrand models exist for Horn clause theories in first-order logic. Building on a previous development of Herbrand models for nominal logic [4], we now show that nominal logic programs also have least Herbrand models.

Theorem 1 (Nominal Herbrand models). *A collection of nominal Horn program clauses is satisfiable in nominal logic if and only if it has an Herbrand model.*

Proof. We note without proof that we can prenex-normalize all $\exists$ and N quantifiers in goals in D-formulas out to the top level as $\forall$ and N quantifiers. Then a collection of normalized D-formulas is a $\mathsf{N}\forall$-theory in the sense of [4], so has a model iff it has an Herbrand model.

Lemma 1. *Let Δ be a set of closed program clauses and $\mathcal{M}$ a nonempty set of Herbrand models of Δ. Then $\mathcal{H} = \bigcap \mathcal{M}$ is also an Herbrand model of Δ.*

Proof. We first note that the intersection of a collection of equivariant sets is still equivariant, so $\mathcal{H}$ is an Herbrand model. To prove it models Δ, we show by mutual induction that

1. For any program clause D, if $\forall M \in \mathcal{M}. M \vDash D$ then $\mathcal{H} \vDash D$; and
2. For any goal formula G, if $\mathcal{H} \vDash G$ then $\forall M \in \mathcal{M}. M \vDash G$.

All the cases are standard except for $\mathsf{N}\mathsf{a}.G$ and $\mathsf{N}\mathsf{a}.D$. If $\forall M \in \mathcal{M}. M \vDash \mathsf{N}\mathsf{a}.D$ then for each M, $M \vDash (\mathsf{b}\ \mathsf{a}) \cdot D$ for any fresh b not in $supp(\mathsf{N}\mathsf{a}.D)$. Without loss of generality we can choose a $\mathsf{b} \notin supp(\mathsf{N}\mathsf{a}.D)$ such that $\forall M \in \mathcal{M}. M \vDash (\mathsf{b}\ \mathsf{a}) \cdot D$. Appealing to the induction hypothesis, we obtain $\mathcal{H} \vDash (\mathsf{b}\ \mathsf{a}) \cdot D$ whence $\mathcal{H} \vDash \mathsf{N}\mathsf{a}.D$. The case for $\mathsf{N}\mathsf{a}.G$ is similar.

An immediate consequence is that a $\subseteq$-least Herbrand model $\mathcal{H}_\Delta = \bigcap\{\mathcal{H} \mid \mathcal{H} \vDash \Delta\}$ exists for any nominal Horn theory Δ. Moreover, $\mathcal{H}_\Delta$ consists of all ground atoms entailed by Δ, as we now show.

Theorem 2. *Let Δ be a set of program clauses. Then $\mathcal{H}_\Delta = \{A \in B_{\mathcal{L}} \mid \Delta \vDash A\}$.*

Proof. If $A \in \mathcal{H}_\Delta$, then A is valid in every Herbrand model of Δ, so by Theorem 1, A is valid in every model of Δ. Conversely, if $\Delta \vDash A$ then since $\mathcal{H}_\Delta \vDash \Delta$ we have $\mathcal{H}_\Delta \vDash A$; thus $A \in \mathcal{H}_\Delta$.

3.2 Fixed Point Semantics

Classical fixed point theorems assert the existence of a fixed point. However, to ensure that the fixed point of an operator on Herbrand models is still an Herbrand model we need an additional constraint: we require that the operator is also equivariant, in the following sense.

Definition 1. *A set operator* $\tau : \mathcal{P}(B_\mathcal{L}) \to \mathcal{P}(B_\mathcal{L})$ *is called* equivariant *if* $(\mathsf{a}\ \mathsf{b}) \cdot \tau(S) = \tau((\mathsf{a}\ \mathsf{b}) \cdot S)$.

Theorem 3. *Suppose* $\tau : \mathcal{P}(B_\mathcal{L}) \to \mathcal{P}(B_\mathcal{L})$ *is equivariant and monotone. Then* $lfp(\tau) = \bigcap\{S \in \mathcal{P}(B_\mathcal{L}) \mid \tau(S) \subseteq S\}$ *is the least fixed point of* τ *and is equivariant. If, in addition,* τ *is continuous, then* $lfp(\tau) = \tau^\omega = \bigcup_{i=0}^{\omega} \tau^i(\varnothing)$.

Proof. By the Knaster-Tarski fixed-point theorem, $lfp(\tau)$ is the least fixed point of τ. We must show that it is equivariant. It suffices to show that if $A \in lfp(\tau)$ then $(a\ b) \cdot A \in lfp(\tau)$. Assume $A \in lfp(\tau)$. Then for any pre-fixed point S of τ (satisfying $\tau(S) \subseteq S$), we have $A \in S$. Let S be an arbitrary pre-fixed point of τ. Note that $\tau((a\ b) \cdot S) = (a\ b) \cdot \tau(S) \subseteq (a\ b) \cdot S$, so $(a\ b) \cdot S$ is also a pre-fixed point of τ. Hence $A \in (a\ b) \cdot S$ so $(a\ b) \cdot A \in (a\ b) \cdot (a\ b) \cdot S = S$.

The second part follows immediately from Kleene's fixed point theorem.

Definition 2. *Let* S *be an Herbrand interpretation and* D *a program clause. Define the transformation* $\tau_D : \mathcal{P}(B_\mathcal{L}) \to \mathcal{P}(B_\mathcal{L})$ *by*

$$\begin{aligned} \tau_\top(S) &= S & \tau_A(S) &= S \cup A \\ \tau_{D_1 \wedge D_2}(S) &= \tau_{D_1}(S) \cup \tau_{D_2}(S) & \tau_{\forall X:\sigma.D} &= \textstyle\bigcup_{t:\sigma} \tau_{D[t/x]}(S) \\ \tau_{G \supset D} &= \begin{cases} \tau_D(S) & (S \vDash G) \\ S & otherwise \end{cases} & \tau_{\text{И}\mathsf{a}:\nu.D} &= \textstyle\bigcup_{\mathsf{b}:\nu \notin supp(\text{И}\mathsf{a}.D)} \tau_{(\mathsf{a}\ \mathsf{b})\cdot D}(S) \end{aligned}$$

We define τ_Δ as $\tau_{D_1 \wedge \cdots \wedge D_n}$ provided $\Delta = \{D_1, \ldots, D_n\}$ and each D_i is closed.

Lemma 2. *For any program* Δ*,* τ_Δ *is monotone and continuous.*

Proof. We prove by induction on the structure of D that τ_D has the above properties. Monotonicity is straightforward. For continuity, the cases for $\top, \wedge, \forall,$ И, and atomic formulas are straightforward. For $G \supset D$, let $S_0, S_1, \ldots,$ be an ω-chain of subsets of $B_\mathcal{L}$. Suppose that $A \in \tau_{G\supset D}(\bigcup_i S_i)$. If $\bigcup_i S_i \vDash G$ then $A \in \tau_D(\bigcup_i S_i)$, and by induction $A \in \bigcup_i(\tau_D(S_i)) = \bigcup_i \tau_{G\supset D}(S_i)$. Otherwise, $A \in \bigcup_i(S_i) = \bigcup_i \tau_{G\supset D}(S_i)$. This shows that $\tau_{G\supset D}(\bigcup_i S_i) \subseteq \bigcup_i \tau_{G\supset D}(S_i)$. For the reverse direction, suppose $A \in \bigcup_i \tau_{G\supset D}(S_i)$. Then for some i, $A \in \tau_{G\supset D}(S_i)$. There are two cases. If $S_i \vDash G$, then $A \in \tau_D(S_i) = \tau_{G\supset D}(S_i) \subseteq \tau_{G\supset D}(\bigcup_i(S_i))$. Otherwise, $A \in S_i = \tau_{G\supset D}(S_i) \subseteq \tau_{G\supset D}(\bigcup_i(S_i))$.

Lemma 3. *For any* $\mathsf{a}, \mathsf{b} \in \mathbb{A}$*,* $(\mathsf{a}\ \mathsf{b}) \cdot \tau_D(S) = \tau_{(\mathsf{a}\ \mathsf{b})\cdot D}((\mathsf{a}\ \mathsf{b}) \cdot S)$*. In particular, if* Δ *is a closed program with* $FV(\Delta) = supp(\Delta) = \varnothing$*, then* τ_Δ *is equivariant.*

Proof. The proof is by induction on the structure of D. The cases for $\top, A, \wedge$ are straightforward; for $\supset$ we need the easy observation that $S \vDash G \iff (\mathsf{a}\ \mathsf{b}) \cdot S \vDash (\mathsf{a}\ \mathsf{b}) \cdot G$. For $\forall X{:}\sigma.D$ formulas, the proof is straightforward once we observe that $(\mathsf{a}\ \mathsf{b}) \cdot \bigcup_{t:\sigma} \tau_{D[t/X]}(S) = \bigcup_{t:\sigma} \tau_{((\mathsf{a}\ \mathsf{b})\cdot D)[(\mathsf{a}\ \mathsf{b})\cdot t/X]}((\mathsf{a}\ \mathsf{b}) \cdot S) = \bigcup_{u:\sigma} \tau_{((\mathsf{a}\ \mathsf{b})\cdot D)[u/X]}((\mathsf{a}\ \mathsf{b}) \cdot S) = \tau_{(\mathsf{a}\ \mathsf{b})\cdot \forall X.D}((\mathsf{a}\ \mathsf{b}) \cdot S)$. For И, the argument is similar.

Theorem 4. $\mathcal{H}_\Delta = lfp(\tau_\Delta) = \tau_\Delta^\omega$.

Proof. Clearly $\tau_\Delta^\omega = lfp(\tau_\Delta)$ by Theorem 3. We show that $\mathcal{H}_\Delta \subseteq \tau_\Delta^\omega$ and $lfp(\tau_\Delta) \subseteq \mathcal{H}_\Delta$.

For $\mathcal{H}_\Delta \subseteq \tau_\Delta^\omega$, it suffices to show that τ_Δ^ω is a model of Δ. We prove by induction on D that if $\tau_D(\mathcal{M}) = \mathcal{M}$ then $\mathcal{M} \vDash D$; from this it follows that $\tau_\Delta^\omega \vDash D$ for each $D \in \Delta$. If $D = \top$, trivially $\mathcal{M} \vDash \top$. If $D = A$, then clearly $\mathcal{M} \cup \{A\} = \tau_A(\mathcal{M}) = \mathcal{M}$ implies $A \in \mathcal{M}$ so $\mathcal{M} \vDash A$. If $D = D_1 \wedge D_2$, then $\tau_{D_1 \wedge D_2}(\mathcal{M}) = \tau_{D_1}(\mathcal{M}) \cup \tau_{D_2}(\mathcal{M}) = \mathcal{M}$ implies $\tau_{D_1}(\mathcal{M}) = \tau_{D_2}(\mathcal{M}) = \mathcal{M}$ since τ_{D_1}, τ_{D_2} are monotone. Then using the induction hypothesis $\mathcal{M} \vDash D_1 \wedge D_2$. If $D = G \supset D'$, we assume that $\mathcal{M} \vDash G$. Then $\tau_{G \supset D'}(\mathcal{M}) = \tau_{D'}(\mathcal{M}) = \mathcal{M}$ so by induction $\mathcal{M} \vDash D'$. Hence $\mathcal{M} \vDash G \supset D'$. For $D = \forall X{:}\sigma.D'$, note that $\mathcal{M} = \tau_{\forall X.D'}(\mathcal{M}) = \bigcup_{t:\sigma} \tau_{D'[t/X]}(\mathcal{M})$ implies $\tau_{D'[t/X]}(\mathcal{M}) = \mathcal{M}$ for every $t : \sigma$. Hence by the induction hypothesis $\mathcal{M} \vDash D'[t/X]$ for every $t : \sigma$; consequently $M \vDash \forall X.D'$. For $D = \text{И}\mathsf{a}{:}\nu.D'$, note that $\mathcal{M} = \tau_{\text{И}\mathsf{a}.D}(\mathcal{M}) = \bigcup_{\mathsf{b}:\nu \notin supp(\text{И}\mathsf{a}.D)} \tau_{(\mathsf{a}\ \mathsf{b})\cdot D'}(\mathcal{M})$ implies $\tau_{(\mathsf{a}\ \mathsf{b})\cdot D'}(\mathcal{M}) = \mathcal{M}$ for every fresh b. Hence by the induction hypothesis $\mathcal{M} \vDash (\mathsf{a}\ \mathsf{b}) \cdot D'$ for every fresh b; consequently $M \vDash \text{И}\mathsf{a}.D'$.

For $lfp(\tau_\Delta) \subseteq \mathcal{H}_\Delta$, it suffices to show that $\mathcal{H}_\Delta$ is a pre-fixed point of τ_Δ, that is, $\tau_\Delta(\mathcal{H}_\Delta) \subseteq \mathcal{H}_\Delta$. We prove that for any D, if $\mathcal{H}_\Delta \vDash D$ then $\tau_D(\mathcal{H}_\Delta) \subseteq \mathcal{H}_\Delta$, by induction on the structure of D. For $D = \top$, clearly $\tau_\top(\mathcal{H}_\Delta) = \mathcal{H}_\Delta$. If $D = D_1 \wedge D_2$, then $\tau_{D_1 \wedge D_2}(\mathcal{H}_\Delta) = \tau_{D_1}(\mathcal{H}_\Delta) \cup \tau_{D_2}(\mathcal{H}_\Delta) \subseteq \mathcal{H}_\Delta$ since $\tau_{D_i}(\mathcal{H}_\Delta) \subseteq \mathcal{H}_\Delta$ by induction for $i = 1, 2$. The case for A follows from the fact that since $\mathcal{H}_\Delta \vDash A$, $\tau_A(\mathcal{H}_\Delta) = \mathcal{H}_\Delta \cup \{A\} = \mathcal{H}_\Delta$. The case for implication follows by case analysis: since by assumption $\mathcal{H}_\Delta \vDash G \supset D$, if $\mathcal{H}_\Delta \vDash G$, then $\mathcal{H}_\Delta \vDash D$, and by induction $\tau_{G \supset D}(\mathcal{H}_\Delta) = \tau_D(\mathcal{H}_\Delta) \subseteq \mathcal{H}_\Delta$; however, if $\mathcal{H}_\Delta \nvDash G$, then $\tau_{G \supset D}(\mathcal{H}_\Delta) = \mathcal{H}_\Delta$. For D of the form $\forall X{:}\sigma.D$, by induction $\tau_{D[t/X]}(\mathcal{H}_\Delta) \subseteq \mathcal{H}_\Delta$ for any $t : \sigma$ so $\bigcup_{t:\sigma} \tau_{D[t/X]}(\mathcal{H}_\Delta) \subseteq \mathcal{H}_\Delta$. Similarly, for $\text{И}\mathsf{a}{:}\nu.D$, by induction $\tau_{(\mathsf{a}\ \mathsf{b})\cdot D}(\mathcal{H}_\Delta) \subseteq \mathcal{H}_\Delta$ for any $\mathsf{b} \notin supp(\text{И}\mathsf{a}.D)$ so $\bigcup_{\mathsf{b}:\nu \notin supp(\text{И}\mathsf{a}.D)} \tau_{(\mathsf{a}\ \mathsf{b})\cdot D}(\mathcal{H}_\Delta) \subseteq \mathcal{H}_\Delta$.

4 Proof-theoretic semantics

In proof-theoretic semantics, an approach due to Miller, Nadathur, Scedrov, and Pfenning [14], we characterize well-behaved logic programming languages as those for which *goal-directed* proof search is complete. This criterion was made concrete by defining *uniform proofs* as sequent calculus proofs in which every sequent in which the goal formula is non-atomic decomposes the goal. An *abstract logic programming language* is then defined as a collection of goal formulas and program clauses for which uniform proof search is complete. Proof-theoretic semantics has been extended to a variety of settings; most relevant here is work on constraint logic programming in a proof theoretic setting [6, 10].

A uniform proof-theoretic approach to nominal logic programming was investigated by Gabbay and Cheney [8]. However, this approach was unsatisfactory in some respects. First, the underlying sequent calculus used rules for И that were rather complex and that suggested an approach to proof-search for И-formulas quite unlike the intuitive "generate a fresh name and proceed" approach employed in αProlog. Second, the *freshness rule* in NL_{Seq}, corresponding to the nominal logic axiom $\forall \overline{x}.\exists a.a \# x$, is not goal-directed but cannot be delayed past the $\exists R$ and $\forall L$ rules. Instead, it was necessary to weaken the definition of uniform proof in order to permit applications of the freshness principle before these rules.

The first problem has been partially addressed by an alternative sequent calculus for nominal logic called $NL^{\Rightarrow}$ [3], in which the И-quantifier rules take a simpler form. We adopt a variation on the $NL^{\Rightarrow}$ approach that also avoids the second problem: specifically, we define an "amalgamated" proof system that separates the term-level constraint-based reasoning (including the freshness rule) from logical reasoning and proof search. This technique was employed by Darlington and Guo [6] and further developed by Leach et al. [10] in studying the semantics of constraint logic programs.

In this section we introduce the amalgamated proof system and relate it to the model-theoretic semantics in the previous section. We assert without proof that this amalgamated system is sound and complete with respect to an intuitionistic form of nominal logic. We also introduce a second *residuated* proof system that explicates the process of reducing a goal to an answer constraint; this system forms an important link between the proof theory and the operational semantics in the next section. Residuated proof search corresponds to ordinary proof search in a natural way. Here and elsewhere, we use Γ for a set of goal formulas, Δ for a set of program clauses, and ∇ for a set of constraints.

The proof rules in Figure 5 describe a proof system that first proceeds by decomposing the goal to an atomic formula, which is then solved by refining a program clause. This system reflects the behavior of an interpreter for nominal logic programs, such as αProlog. The uniform proof judgment $\Sigma : \Delta; \nabla \Longrightarrow G$ indicates that G is derivable from Δ and ∇ in context Σ, while the focused proof judgment $\Sigma : \Delta; \nabla \xrightarrow{D} G$ indicates that atomic goal A is derivable from Δ and ∇ by refining the program clause D (using Δ to help solve any residual goals). The judgment $\Sigma : \nabla \vDash C$ is the ordinary satisfaction relation defined in Section 2.

These rules are unusual in several important respects. First, the *hyp* rule requires solving a constraint of the form $A \sim B$, which we define as "there exists a permutation π such that $\pi \cdot A \approx B$". In contrast usually the hypothesis rule requires only that $A \approx A'$. Our rule accounts for the fact that equivalent atomic formulas may not be syntactically equal as nominal terms, but only equal modulo a permutation. Second, the proof system treats constraints specially, sep-

$$\frac{\Sigma : \nabla \vDash C}{\Sigma : \Delta; \nabla \Longrightarrow C}\ con \qquad \frac{\Sigma : \Delta; \nabla \Longrightarrow G_1 \quad \Sigma : \Delta; \nabla \Longrightarrow G_2}{\Sigma : \Delta; \nabla \Longrightarrow G_1 \wedge G_2}\ \wedge R$$

$$\frac{\Sigma : \Delta; \nabla \Longrightarrow G_i}{\Sigma : \Delta; \nabla \Longrightarrow G_1 \vee G_2}\ \vee R \qquad \frac{\Sigma : \nabla \vDash \exists x.C \quad \Sigma, x : \Delta; \nabla, C \Longrightarrow G}{\Sigma : \Delta; \nabla \Longrightarrow \exists x^\sigma.G}\ \exists R$$

$$\frac{\Sigma : \nabla \vDash \text{И}\mathsf{a}.C \quad \Sigma\#\mathsf{a} : \Delta; \nabla, C \Longrightarrow G}{\Sigma : \Delta; \nabla \Longrightarrow \text{И}\mathsf{a}.G}\ \text{И}R \qquad \frac{\Sigma : \Delta; \nabla \xrightarrow{D} A \quad D \in \Delta}{\Sigma : \Delta; \nabla \Longrightarrow A}\ sel$$

$$\frac{\Sigma : \nabla \vDash A' \sim A}{\Sigma : \Delta; \nabla \xrightarrow{A'} A}\ hyp \qquad \frac{\Sigma : \Delta; \nabla \xrightarrow{D_i} A}{\Sigma : \Delta; \nabla \xrightarrow{D_1 \wedge D_2} A}\ \wedge L \qquad \frac{\Sigma : \Delta; \nabla \xrightarrow{D} A \quad \Sigma : \Delta; \nabla \Longrightarrow G}{\Sigma : \Delta; \nabla \xrightarrow{G \supset D} A}\ \supset L$$

$$\frac{\Sigma : \nabla \vDash \exists x.C \quad \Sigma, x : \Delta; \nabla, C \xrightarrow{D} A}{\Sigma : \Delta; \nabla \xrightarrow{\forall x^\sigma.D} A}\ \forall L \qquad \frac{\Sigma : \nabla \vDash \text{И}\mathsf{a}.C \quad \Sigma\#\mathsf{a} : \Delta; \nabla, C \xrightarrow{D} A}{\Sigma : \Delta; \nabla \xrightarrow{\text{И}\mathsf{a}.D} A}\ \text{И}L$$

Fig. 5. Uniform/focused proof search for intuitionistic nominal logic

arating them into a context ∇. This is necessary because the role of constraints is quite different from that of program clauses: the former are used exclusively for constraint solving whereas the latter are used in backchaining. Third, the $\text{И}L, \text{И}R, \exists R$ and $\forall R$ rules are permitted to introduce a constraint on the witness name a or variable X rather than providing a witness term. This constraint-based treatment makes it possible to compartmentalize all reasoning about the constraint domain (including the freshness rule) in the judgment $\Sigma : \nabla \vDash C$.

For example, the goal $\text{И}\mathsf{a}.\exists X.\mathsf{a} \# X$ has the following uniform derivation:

$$\cfrac{\cfrac{\Sigma\#\mathsf{a} : \nabla \vDash \exists X.\mathsf{a} \# X \quad \Sigma\#\mathsf{a}, X : \Delta; \nabla \Longrightarrow \exists X.\mathsf{a} \# X}{\Sigma\#\mathsf{a} : \Delta; \nabla \Longrightarrow \exists X.\mathsf{a} \# X}\ \exists R}{\Sigma : \Delta; \nabla \Longrightarrow \text{И}\mathsf{a}.\exists X.\mathsf{a} \# X}\ \text{И}R$$

where $\Sigma\#\mathsf{a} : \nabla \vDash \exists X.\mathsf{a} \# X$ is clearly valid for any ∇ (take X to be any ground name besides a). In contrast, in previous proof-theoretic approaches, proof search for $\text{И}\mathsf{a}.\exists X.\mathsf{a} \# X$ requires using the freshness rule before $\exists R$.

We first show (by induction on derivations) that the restricted system is sound with respect to the denotational semantics.

Theorem 5 (Soundness). *If $\Sigma : \Delta; \nabla \Longrightarrow G$ is derivable then $\Sigma : \Delta, \nabla \vDash G$. Similarly, if $\Sigma : \Delta; \nabla \xrightarrow{D} G$ is derivable then $\Sigma : \Delta, D, \nabla \vDash G$.*

We next show a restricted form of completeness. Since the denotational semantics is classical while the proof theory is constructive, it is too much to expect that classical completeness holds. For example, $A, B : \cdot \vDash A \approx B \vee A \# B$ is valid, but $A, B : \cdot; \cdot \Longrightarrow A \approx B \vee A \# B$ is not derivable (and indeed not intuitionistically valid). Instead, however, we can prove that any valuation θ that satisfies G also satisfies a constraint which proves G.

Proposition 1. *For any $\Sigma, \Delta, G, D, i \geq 0$:*

$$\frac{}{\Sigma : \Delta \Longrightarrow C \setminus C}\,con \qquad \frac{\Sigma : \Delta \Longrightarrow G_1 \setminus C_1 \quad \Sigma : \Delta \Longrightarrow G_2 \setminus C_2}{\Sigma : \Delta \Longrightarrow G_1 \wedge G_2 \setminus C_1 \wedge C_2}\,\wedge R$$

$$\frac{\Sigma : \Delta \Longrightarrow G_i \setminus C}{\Sigma : \Delta \Longrightarrow G_1 \vee G_2 \setminus C}\,\vee R \qquad \frac{\Sigma, x : \Delta \Longrightarrow G \setminus C}{\Sigma : \Delta \Longrightarrow \exists x^\sigma.G \setminus \exists x.C}\,\exists R$$

$$\frac{\Sigma \# \mathsf{a} : \Delta \Longrightarrow G \setminus C}{\Sigma : \Delta \Longrightarrow \text{И}\mathsf{a}.G \setminus \text{И}\mathsf{a}.C}\,\text{И}R \qquad \frac{\Sigma : \Delta \xrightarrow{D} A \setminus G \quad (D \in \Delta) \quad \Sigma : \Delta \Longrightarrow G \setminus C}{\Sigma : \Delta \Longrightarrow A \setminus C}\,back$$

$$\frac{}{\Sigma : \Delta \xrightarrow{A'} A \setminus A \sim A'}\,hyp \qquad \frac{\Sigma : \Delta \xrightarrow{D_i} A \setminus G}{\Sigma : \Delta \xrightarrow{D_1 \wedge D_2} A \setminus G}\,\wedge L \qquad \frac{\Sigma : \Delta \xrightarrow{D} A \setminus G'}{\Sigma : \Delta \xrightarrow{G \supset D} A \setminus G \wedge G'}\,\supset L$$

$$\frac{\Sigma, x : \Delta \xrightarrow{D} A \setminus G}{\Sigma : \Delta \xrightarrow{\forall x^\sigma.D} A \setminus \exists x.G}\,\forall L \qquad \frac{\Sigma \# \mathsf{a} : \Delta; \nabla, C \xrightarrow{D} A}{\Sigma : \Delta; \nabla \xrightarrow{\text{И}\mathsf{a}.D} A}\,\text{И}L$$

Fig. 6. Residuated uniform/focused proof search

1. *If $\Sigma : \tau^i_\Delta, \theta \vDash G$ then for some ∇, $\Sigma : \theta \vDash \nabla$ and $\Sigma : \Delta; \nabla \Longrightarrow G$ is derivable.*
2. *If $\Sigma : \tau_D(\tau^i_\Delta), \theta \vDash A$ but $\Sigma : \tau^i_\Delta, \theta \nvDash A$ then for some ∇, $\Sigma : \theta \vDash \nabla$ and $\Sigma : \Delta; \nabla \xrightarrow{D} A$.*

Proof. For the first part, proof is by induction on i and G; most cases are straightforward. For atomic goals A, i must be > 0, and if $\Sigma : \tau^{i-1}_\Delta; \theta \vDash A$ then we use part (1) of the induction hypothesis with $i-1$; otherwise $\Sigma : \tau^{i-1}_\Delta; \theta \nvDash A$ so we unwind τ^i_Δ to $\tau_D(\tau^{i-1}_\Delta)$ for some $D \in \Delta$, apply part (2), and use rule *sel*. Similarly, the second part follows by induction on i and D, unwinding the definition of τ_D in each case.

Theorem 6 (Algebraic Completeness). *If $\Sigma : \Delta, \theta \vDash G$ then there exists a constraint ∇ such that $\Sigma : \Delta, \theta \vDash \nabla$ and $\Sigma : \Delta; \nabla \Longrightarrow G$ is derivable.*

Proof. If $\Sigma : \Delta, \theta \vDash G$, then there is some n such that $\Sigma : \tau^n_\Delta, \theta \vDash G$, so Proposition 1 applies.

We can also extend this to a "logical" completeness result, namely that if an answer C classically implies G, then there is a finite set of constraints which prove G and whose disjunction covers C.

Theorem 7 (Logical Completeness). *If $\Sigma : \Delta, C \vDash G$ then there exists a finite set of constraints $C_1, \ldots, C_n$ such that $\Sigma : C \vDash C_1 \vee \cdots \vee C_n$ and for each i, $\Sigma : \Delta; C_i \Longrightarrow G$.*

The rules in Figure 5 have the potential disadvantage that an arbitrary constraint C is allowed in the rules $\exists R, \forall R, \text{И}L, \text{И}R$. Figure 6 shows a refined form of these derivations that avoids this nondeterminism. Specifically, the judgment $\Sigma : \Delta \Longrightarrow G \setminus C$ means that given context Σ and program Δ, goal G reduces to constraint C; similarly, $\Sigma : \Delta \xrightarrow{D} A \setminus G$ means that goal formula G suffices to prove A from D. We state without proof the following appropriate soundness and completeness properties; their proofs are straightforward inductions.

Table 1. Operational semantics rules for nominal logic programs

	$\Sigma\langle\Gamma \mid \nabla\rangle \longrightarrow \Sigma'\langle\Gamma' \mid \nabla'\rangle$	provided
B	$\Sigma\langle A, \Gamma \mid \nabla\rangle \longrightarrow \Sigma\langle G, \Gamma \mid \nabla\rangle$	$\Sigma : \Delta \xrightarrow{D} A \setminus G, D \in \Delta$
$\top$	$\Sigma\langle\top, \Gamma \mid \nabla\rangle \longrightarrow \Sigma\langle\Gamma \mid \nabla\rangle$	
$\wedge$	$\Sigma\langle G_1 \wedge G_2, \Gamma \mid \nabla\rangle \longrightarrow \Sigma\langle G_1, G_2, \Gamma \mid \nabla\rangle$	
$\vee$	$\Sigma\langle G_1 \vee G_2, \Gamma \mid \nabla\rangle \longrightarrow \Sigma\langle G_i, \Gamma \mid \nabla\rangle$	($i = 1$ or 2)
$\exists$	$\Sigma\langle\exists X{:}\sigma.G, \Gamma \mid \nabla\rangle \longrightarrow \Sigma, X{:}\sigma\langle G, \Gamma \mid \nabla\rangle$	
И	$\Sigma\langle \text{И}\mathsf{a}{:}\nu.G, \Gamma \mid \nabla\rangle \longrightarrow \Sigma, \mathsf{a}{:}\nu\langle G, \Gamma \mid \nabla\rangle$	
C	$\Sigma\langle C, \Gamma \mid \nabla\rangle \longrightarrow \Sigma\langle\Gamma \mid \nabla, C\rangle$	∇, C consistent

Theorem 8 (Residuated Soundness).

1. *If $\Sigma : \Delta \Longrightarrow G \setminus C$ then $\Sigma : \Delta; C \Longrightarrow G$.*
2. *If $\Sigma : \Delta; \nabla \Longrightarrow G$ and $\Sigma : \Delta \xrightarrow{D} A \setminus G$ then $\Sigma : \Delta; \nabla \xrightarrow{D} A$.*

Theorem 9 (Residuated Completeness).

1. *If $\Sigma : \Delta; \nabla \Longrightarrow G$ then there exists a constraint C such that $\Sigma : \Delta \Longrightarrow G \setminus C$ and $\Sigma : \nabla \vDash C$.*
2. *If $\Sigma : \Delta; \nabla \xrightarrow{D} A$ then there exists goal G and constraint C such that $\Sigma : \Delta \xrightarrow{D} A \setminus G$ and $\Sigma : \Delta \Longrightarrow G \setminus C$ and $\Sigma : \nabla \vDash C$.*

5 Operational Semantics

We now give a CLP-style operational semantics for nominal logic programs. The rules of the operational semantics are shown in Table 1. A program state is a triple of the form $\Sigma\langle\Gamma \mid \nabla\rangle$. Note that the backchaining step is defined in terms of residuated focused proof, $\Sigma : \Delta \xrightarrow{D} A \setminus G$.

We now state the operational soundness and completeness properties. The proofs are straightforward by cases or induction, so omitted. To simplify notation, we write $\Sigma : \Delta \Longrightarrow \overline{\Gamma} \setminus \overline{C}$ where $\Gamma = \{G_1, \ldots, G_n\}$ and $\overline{C} = \{C_1, \ldots, C_n\}$ to abbreviate $\Sigma : \Delta \Longrightarrow G_1 \setminus C_1, \ldots, \Sigma : \Delta \Longrightarrow G_n \setminus C_n$; also, if $\Sigma\langle G \mid \varnothing\rangle \longrightarrow^* \Sigma; \Sigma'\langle\varnothing \mid C\rangle$, then we abbreviate this as $\Sigma\langle G\rangle \Downarrow \exists\Sigma'[C]$.

Proposition 2. *If $\Sigma\langle\Gamma \mid \nabla\rangle \longrightarrow \Sigma'\langle\Gamma' \mid \nabla'\rangle$ and $\Sigma' : \Delta \Longrightarrow \overline{\Gamma'} \setminus \overline{C'}$ then there exist $\overline{C}$ such that $\Sigma : \Delta \Longrightarrow \overline{\Gamma} \setminus \overline{C}$ and $\Sigma' : \nabla', \overline{C'} \vDash \nabla, \overline{C}$.*

Corollary 1. *If $\Sigma\langle G\rangle \Downarrow \nabla$ then:*

1. *there exists C such that $\Sigma : \nabla \vDash C$ and $\Sigma : \Delta \Longrightarrow G \setminus C$;*
2. *$\Sigma' : \Delta; \nabla \Longrightarrow G$; and*
3. *$\Sigma' : \Delta, \nabla \vDash G$.*

Proposition 3. *For any Γ and satisfiable ∇, $\overline{C}$, if $\Sigma : \Delta \Longrightarrow \overline{\Gamma} \setminus \overline{C}$ then for some Σ', ∇', and $\overline{C'}$ we have $\Sigma\langle\Gamma \mid \nabla\rangle \longrightarrow \Sigma'\langle\Gamma' \mid \nabla'\rangle$, $\Sigma' : \Delta \Longrightarrow \overline{\Gamma'} \setminus \overline{C'}$, and $\Sigma' : \nabla, \overline{C} \vDash \nabla', \overline{C'}$.*

Corollary 2. *1. If $\Sigma : \Delta \Longrightarrow G \setminus C$ and C is satisfiable then for some ∇, we have $\Sigma\langle G\rangle \Downarrow \nabla$ and $\Sigma : C \vDash \nabla$.*
2. If $\Sigma : \Delta; \nabla \Longrightarrow G$ and ∇ is satisfiable then for some ∇', we have $\Sigma\langle G\rangle \Downarrow \nabla'$ and $\Sigma : \nabla \vDash \nabla'$.
3. If $\Sigma : \Delta, \theta \vDash G$ then for some ∇, we have $\Sigma\langle G\rangle \Downarrow \nabla$ and $\Sigma : \theta \vDash \nabla$.
4. If $\Sigma : \Delta, C \vDash G$ then for some $\overline{\nabla}$, we have $\Sigma\langle G\rangle \Downarrow \overline{\nabla}$ and $\Sigma : C \vDash \nabla_1 \vee \cdots \vee \nabla_n$.

6 Applications

Correctness of elaboration In an implementation, program clauses are often *elaborated* into a normal form $\forall\Sigma[G \supset A]$ which is easier to manipulate and optimize. We define the elaboration of a program clause or program as the result of normalizing it with respect to the following confluent, terminating rewrite system:

$$\begin{array}{ll}
G \supset G' \supset D \to G \wedge G' \supset D & G \supset D \wedge D' \to (G \supset D) \wedge (G \supset D') \\
G \supset \forall x.D \to \forall x.(G \supset D)\ (x \notin FV(G)) & G \supset \text{И}\mathsf{a}.D \to \text{И}\mathsf{a}.(G \supset D)\ (\mathsf{a} \notin supp(G)) \\
\forall x.(D \wedge D') \to \forall x.D \wedge \forall x.D' & \text{И}\mathsf{a}.(D \wedge D') \to \text{И}\mathsf{a}.D \wedge \text{И}\mathsf{a}.D'
\end{array}$$

$$\begin{array}{lll}
G \supset \top \to \top & D \wedge \top \to \top & \top \wedge D \to \top \\
\forall x.\top \to \top & \text{И}\mathsf{a}.\top \to \top & \Delta, D \wedge D' \to \Delta, D, D'
\end{array}$$

It is straightforward to show that elaborated programs consist only of closed formulas $Q_1 \cdots Q_n[G \supset A]$, where $\overline{Q}$ is a sequence of $\forall$- or И-quantifiers. Moreover, this translation preserves the meaning of the program:

Theorem 10. *1. If $\Delta \to \Delta'$ then $\Sigma : \Delta; \nabla \Longrightarrow G$ iff $\Sigma : \Delta'; \nabla \Longrightarrow G$.*
2. If $\Delta \to \Delta'$ then $\Sigma : \Delta; \nabla \xrightarrow{D} A$ iff $\Sigma : \Delta'; D \Longrightarrow A$.
3. If $D \to D'$ hen $\Sigma : \Delta; \nabla \xrightarrow{D} A$ iff $\Sigma : \Delta; D' \Longrightarrow A$.

A tractable constraint sublanguage In general, nominal constraint solving is NP-complete. However, Urban, Pitts, and Gabbay's nominal unification algorithm solves a tractable special case:

Definition 3. *A nominal constraint C is* simple *if in every subterm of the form $a \# t$, $\langle a\rangle t$, or $(a\ b) \cdot t$, the terms a and b are ground names.*

For example, $\langle \mathsf{a}\rangle X$ and $(\mathsf{a}\ \mathsf{b}) \cdot Y$ are *simple* , while $\langle A\rangle X$ and $((A\ B) \cdot C\ (A\ C) \cdot B) \cdot C$ are not. Constraint solving for *simple* problems involving only $\approx$ and $\#$ is decidable in polynomial time [16]. Moreover, it is easy to see that the residuated and operational semantics reduce any satisfiable goal to a *simple* constraint.

Unfortunately, due to the fact that atomic formulas are equivalent if they are equal modulo a permutation in nominal logic, this is not enough to ensure that nominal logic programs can be executed efficiently, because constraint solving is NP-complete even for *simple* constraints $A \sim B$. Therefore, it is tempting to replace the *hyp* rule with $hyp_{\approx}$, in which we only need to show $\Sigma : \nabla \vDash A \approx B$ in order to conclude $\Sigma : \Delta; \nabla \xrightarrow{A} B$. We write $\Sigma : \Delta; \nabla \Longrightarrow_{\approx} G$ and $\Sigma : \Delta; \nabla \xrightarrow{D}_{\approx} A$ for uniform or focused proofs in which $hyp_{\approx}$ is used instead of hyp.

It is easy to see that such proofs only require solving tractable *simple* constraints involving $\approx$ or $\#$. However, proof search is incomplete for this system since $\vDash (\mathsf{a}\ \mathsf{b}) \cdot p(\mathsf{a}) \approx p(\mathsf{b})$ while $\nvDash p(\mathsf{b}) \approx p(\mathsf{b})$. Indeed, there is no proof of $\Sigma : \text{И}\mathsf{a}.p(\mathsf{a}); \nabla \Longrightarrow_{\approx} \text{И}\mathsf{a}.p(\mathsf{a})$.

There are, however, sublanguages of *simple* nominal Horn clauses for which proof search based on $\approx$ is complete. One interesting example identified by Urban and Cheney [17]. In this language, И is forbidden in goal formulas and program clauses are required to satisfy a well-formedness condition which ensures that every derivation using hyp can be transformed to one using $hyp_{\approx}$. That well-formedness condition is somewhat complicated so we omit further discussion.

We say that a program clause is *И-clause-free* if it has no subformula of the form $\text{И}\mathsf{a}.D$. However, И-quantified goals $\text{И}\mathsf{a}.G$ are allowed. We believe that all the programs considered well-formed according to Urban and Cheney's paper are equivalent to *И-clause-free* programs and vice versa. Careful investigation of this relationship is deferred to the full version of this paper.

To prove the completeness of $\approx$-backchaining for *И-clause-free* programs, we adopt an approach similar to that in Urban and Cheney's proof.

Lemma 4. *Let π be a type-preserving permutation of names in Σ.*

1. *If $\Sigma : \Delta; \nabla \Longrightarrow_{\approx} G$ then $\Sigma : \Delta; \nabla \Longrightarrow_{\approx} \pi \cdot G$.*
2. *If $\Sigma : \Delta; \nabla \xrightarrow{D}_{\approx} A$ then $\Sigma : \Delta; \nabla \xrightarrow{\pi \cdot D}_{\approx} \pi \cdot A$.*

Proof. By induction on derivations. The cases for quantifiers require some care; for example, in the case for $\forall x.G[x]$, we have subderivation $\Sigma, x : \Delta; \nabla, C[x] \Longrightarrow_{\approx} \pi \cdot G[\pi \cdot x]$ where $\Sigma : \nabla \vDash \exists x.C[x]$. Since π is invertible, we can substitute $y = \pi \cdot x$ to obtain $\Sigma, x : \Delta; \nabla, C[\pi^{-1} \cdot y] \Longrightarrow_{\approx} \pi \cdot G[y]$; moreover, clearly, $\Sigma : \nabla \vDash \exists y.C[\pi^{-1} \cdot y]$, so we can conclude $\Sigma : \Delta; \nabla \Longrightarrow_{\approx} \pi \cdot \forall x.G$.

Theorem 11. *If Δ is И-clause-free then*

1. *If $\Sigma : \Delta; \nabla \Longrightarrow G$ is derivable, $\Sigma : \Delta; \nabla \Longrightarrow_{\approx} G$ is derivable.*
2. *If $\Sigma : \Delta; \nabla \xrightarrow{D} A$ is derivable, there exists a π such that $\Sigma : \Delta; \nabla \xrightarrow{\pi \cdot D}_{\approx} A$ is derivable.*

Proof. The proof is by induction on derivations. The most interesting cases are hyp and sel. For hyp, from $\Sigma : \Delta; \nabla \xrightarrow{\pi \cdot A} A'$ we have $\Sigma : \nabla \vDash A' \sim A$, which by definition means there exists a π such that $\Sigma : \nabla \vDash \pi \cdot A' \approx A$, so $\Sigma : \Delta; \nabla \xrightarrow{\pi \cdot A}_{\approx} A'$. For sel, from $\Sigma : \Delta; \nabla \Longrightarrow A$ we obtain $\Sigma : \Delta; \nabla \xrightarrow{D} A$ for some closed $D \in \Delta$. By induction, for some π, $\Sigma : \Delta; \nabla \xrightarrow{\pi \cdot D}_{\approx} A$. However, since D is closed, $\pi \cdot D = D \in \Delta$ so we may conclude $\Sigma : \Delta; \nabla \Longrightarrow_{\approx} A$ using the sel rule again. Other cases follow by arguments similar to those for Lemma 4. The case for $\text{И}L$ is vacuously true: no instance of $\text{И}L$ can occur in a *И-clause-free* program derivation.

Note that Theorem 11 fails if $\reflectbox{N}L$ is allowed: specifically, given an instance of $\reflectbox{N}L$ deriving $\Sigma : \Delta; \nabla \xrightarrow{\reflectbox{N}\mathsf{a}.D} A$ from $\Sigma\#\mathsf{a} : \Delta; \nabla \xrightarrow{D} A$, we can obtain a derivation of $\Sigma\#\mathsf{a} : \Delta; \nabla \xrightarrow{\pi\cdot D}_{\approx} A$ by induction, but since π may mention a, it is not possible in general to conclude $\Sigma : \Delta; \nabla \xrightarrow{\pi\cdot\reflectbox{N}\mathsf{a}.D}_{\approx} A$.

7 Related and Future Work

Basic soundness and algebraic/logical completeness results similar to those of Jaffar et al. [9] were first proved in [5, Chapter 6]. Although many of the basic ideas are the same, we believe the definitions and proofs in this paper are cleaner and more extensible. For example, extending the proof-theoretic semantics to hereditary Harrop programs (following [10]), supporting negation-as-failure, or extending the constraint domain appears straightforward. Another interesting direction to pursue is the possibility of compiling nominal logic programs to ordinary CLP programs with an appropriate nominal constraint solver written using constraint handling rules. Since the λ-calculus can be axiomatized as an ordinary equational theory in nominal logic, this suggests the interesting possibility that full higher-order logic programming can be provided as "just another constraint domain", as suggested by Michaylov and Pfenning [12].

Nominal logic programming is similar in expressive power to restricted forms of higher-order logic programming, such as Miller's L_λ [13], a subset of higher-order logic programming in which all subterms are required to be *higher-order patterns*. Nominal terms and higher-order patterns are behaviorally very similar [16], although their syntactic form and semantic motivations appear quite different. We believe that the well-behaved fragment of nominal logic programming investigated in this paper is "essentially" the same as L_λ, except for minor differences such as that goal formulas can be passed as arguments to predicates in L_λ. If this is the case, this is an interesting result, because it implies that nominal logic can provide an appropriate *logical* justification for L_λ, which heretofore has not been established.

8 Conclusions

Nominal logic is an interesting extension of ordinary first-order logic with a constraint and equality theory describing terms with bound names up to α-equivalence. In this paper, we have given three semantics for nominal logic programs: a classical, denotational semantics (based on nominal Herbrand models), a constructive, proof-theoretic semantics, and an operational semantics. Moreover, we have proved appropriate soundness and completeness theorems relating the semantics. These results are important because they provide a criterion for judging the correctness of implementations, and suggest that nominal logic programming can be provided as simply another constraint domain within a full CLP system. In addition, we used the proof-theoretic semantics to develop a clean, syntactic characterization of well-behaved programs for which backchaining using the nominal unification algorithm of [16] is complete for proof search.

References

1. J. Cheney and C. Urban. Alpha-Prolog: A logic programming language with names, binding and alpha-equivalence. In *Proceedings of the 20th International Conference on Logic Programming (ICLP 2004)*, number 3132 in LNCS, pages 269–283, 2004.
2. James Cheney. The complexity of equivariant unification. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *LNCS*, pages 332–344. Springer-Verlag, 2004.
3. James Cheney. A simpler proof theory for nominal logic. In *Proceedings of the 2005 Conference on Foundations of Software Science and Computation Structures (FOSSACS 2005)*, number 3441 in LNCS, pages 379–394. Springer-Verlag, 2005.
4. James Cheney. Completeness and Herbrand theorems for nominal logic. *Journal of Symbolic Logic*, 81(1):299–320, 2006.
5. James R. Cheney. *Nominal Logic Programming*. PhD thesis, Cornell University, Ithaca, NY, August 2004.
6. John Darlington and Yike Guo. Constraint logic programming in the sequent calculus. In *Proceedings of the 1994 Conference on Logic Programming and Automated Reasoning (LPAR 1994)*, volume 822 of *Lecture Notes in Computer Science*, pages 200–214. Springer-Verlag, 1994.
7. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.
8. M. J. Gabbay and J. Cheney. A sequent calculus for nominal logic. In H. Ganzinger, editor, *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 139–148, Turku, Finland, 2004. IEEE.
9. Joxan Jaffar, Michael J. Maher, Kim Marriott, and Peter J. Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1–3):1–46, 1998.
10. Javier Leach, Susan Nieva, and Mario Rodríguez-Artalejo. Constraint logic programming with hereditary Harrop formulas. *Theory and Practice of Logic Programming*, 1(4):409–445, July 2001.
11. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
12. Spiro Michaylov and Frank Pfenning. Higher-order logic programming as constraint logic programming. In *Position Papers for the First Workshop on Principles and Practice of Constraint Programming*, pages 221–229, Newport, Rhode Island, April 1993.
13. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Logic and Computation*, 1(4):497–536, 1991.
14. Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
15. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 183:165–193, 2003.
16. C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.
17. Christian Urban and James Cheney. Avoiding equivariant unification. In *Proceedings of the 2005 Conference on Typed Lambda Calculus and Applications (TLCA 2005)*, number 3461 in LNCS, pages 74–89. Springer-Verlag, 2005.