Logic Programming with Names and Binding

James Cheney

Logic & Semantics Seminar

May 6, 2005
Prologue
Brief history

• History: My involvement in this work started at Cambridge almost exactly 2 years and one dissertation ago.

• Thanks!
• $\alpha$Prolog is a logic programming language based on Pitts’ nominal logic

• and using Urban, Pitts, and Gabbay’s nominal unification algorithm

• FreshML : ML :: $\alpha$Prolog : Prolog (+ types)

• I am going to assume that the audience already has some familiarity with all of the above.
Examples (I)

- A (very tired) example: typechecking.

\[
\begin{align*}
  &x : T \in \Gamma \\
  &\Gamma \vdash x : T \\
  &\Gamma \vdash e : T \rightarrow U \\
  &\Gamma \vdash f : T \\
  &\Gamma \vdash e \ f : U \\
  & (x \notin \Gamma) \quad \Gamma, x : T \vdash e : U \\
  &\Gamma \vdash \lambda x. e : T \rightarrow U
\end{align*}
\]

\[
\begin{align*}
tc(G, \text{var}(X), T) & \quad :- \ \text{mem}((X, T), G). \\
tc(G, \text{app}(E, F), U) & \quad :- \ tc(G, E, \text{arr}(T, U)), \ tc(G, F, T). \\
tc(G, \text{lam}(x \backslash E), \text{arr}(T, U)) & \quad :- \ x \ # \ G, \ tc([(x, T) | G], E, U).
\end{align*}
\]
Examples (II)

- A less trivial example: big step semantics for $\lambda_{ref}$.

\[
\begin{align*}
(a \in \text{Lab}) & \quad \frac{}{\langle M, a \rangle \rightarrow \langle M, a \rangle} \\
\langle M, e_1 \rangle \rightarrow \langle M', a \rangle & \quad \frac{}{\langle M', e_2 \rangle \rightarrow \langle M'', v \rangle} \\
\langle M, e_1 := e_2 \rangle & \rightarrow \langle M''[a := v], () \rangle \\
\langle M, e \rangle \rightarrow \langle M', a \rangle & \quad \frac{}{\langle M, !e \rangle \rightarrow \langle M', M'(a) \rangle} \\
\langle M, e \rangle & \rightarrow \langle M', v \rangle \quad (a \notin \text{dom}(M')) \\
\langle M, \text{ref } e \rangle & \rightarrow \langle M'[a := v], a \rangle
\end{align*}
\]

- Interesting part: last rule requires some/any fresh label for new memory cell
Examples (II)

- A less trivial example: big step semantics for $\lambda_{ref}$.

$$(\text{M}, \text{lab}(A)) \quad \text{eval} \quad (\text{M}, \text{lab}(A)).$$

$$(\text{M}, \text{assign}(E_1, E_2)) \quad \text{eval} \quad (\text{M}_3, \text{unit}) \quad \text{:-} \quad (\text{M}, E_1) \quad \text{eval} \quad (\text{M}_1, \text{lab}(A)),
\quad (\text{M}_1, E_2) \quad \text{eval} \quad (\text{M}_2, V),
\quad \text{update}((A, V), M_2, M_3).$$

$$(\text{M}, \text{deref}(E)) \quad \text{eval} \quad (M', V) \quad \text{:-} \quad (M, E) \quad \text{eval} \quad (M', \text{lab}(A)),
\quad \text{mem}((A, V), M').$$

$$(\text{M}, \text{ref}(E)) \quad \text{eval} \quad ([\{(a, V) | M_1\}, \text{lab}(a)) \quad \text{:-} \quad (M, E) \quad \text{eval} \quad (M_1, V),
\quad a \# M_1.$$}

- Interesting part: last rule constrains $\forall$-quantified name to be sufficiently fresh
Examples (III)

• Another example: closure conversion

\[
\begin{align*}
C[x, \Gamma \vdash x]e & = \pi_1(e) \\
C[y, \Gamma \vdash x]e & = C[\Gamma \vdash x](\pi_2(e)) \quad (x \neq y) \\
C[\Gamma \vdash t_1 t_2]e & = \text{let } c = C[\Gamma \vdash t_1]e \\
& \quad \text{in } (\pi_1(c)) \langle C[\Gamma \vdash t_2]e, \pi_2(c) \rangle \\
C[\Gamma \vdash \lambda x. t]e & = \langle \lambda y. C[x, \Gamma \vdash t]y, e \rangle \quad (x, y \notin \Gamma)
\end{align*}
\]
Examples (III)

• Another example: closure conversion

\[
\begin{align*}
\text{cconv}([x \mid G], \text{var}(x), E) &= \text{pi1}(E). \\
\text{cconv}([y \mid G], \text{var}(x), E) &= \text{cconv}(G, \text{var}(x), \text{pi2}(E)). \\
\text{cconv}(G, \text{app}(T_1, T_2), E) &= \text{let}(\text{cconv}(G, T_1, E), c \backslash \\
&\quad \text{app}(\text{pi1}(\text{var}(c)), \\
&\quad \quad \text{pair}(\text{cconv}(G, T_2, E), \text{pi2}(\text{var}(c))))). \\
\text{cconv}(G, \text{lam}(x \backslash T), E) &= \text{pair}(\text{lam}(y \backslash \text{cconv}([x \mid G], T, \text{var}(y))), E) \\
&: x \# G, y \# G.
\end{align*}
\]

• Note: Functions are unwound to relations in \(\alpha\)Prolog.
Nominal logic programming
Notation

\[ a, b \in \mathbb{A} \quad \text{Atoms/Names} \]
\[ f, g \in \text{FnSym} \quad \text{Term symbols} \]
\[ X, Y \in \text{Var} \quad \text{Variables} \]
\[ a, b, t, u ::= \langle \rangle | \langle t, u \rangle | f(t) | X \quad \text{First-order terms} \]
\[ \quad | \langle a \rangle t | \Pi \cdot t | a \quad \text{Nominal terms} \]
\[ \Pi ::= (a \ b) | \text{id} | \Pi \circ \Pi' | \Pi^{-1} | P \quad \text{Permutations} \]
\[ C ::= t \approx u | a \# t \quad \text{Equality, freshness} \]

Note that this includes permutation terms & variables which are not present in nominal logic proper.
Ground swapping

The result of applying a (ground) permutation \( \Pi \) to a (ground) term is:

\[
\begin{align*}
\Pi \cdot a &= \Pi(a) \\
\Pi \cdot \langle \rangle &= \langle \rangle \\
\Pi \cdot \langle t, u \rangle &= \langle \Pi \cdot t, \Pi \cdot u \rangle \\
\Pi \cdot f(t) &= f(\Pi \cdot t) \\
\Pi \cdot \langle b \rangle t &= \langle \Pi \cdot b \rangle \Pi \cdot t
\end{align*}
\]

where

\[
\begin{align*}
id(a) &= a \\
\Pi \circ \Pi'(a) &= \Pi(\Pi'(a)) \\
(a \ b)(c) &= \begin{cases} 
  b & (a = c) \\
  a & (b = c) \\
  c & (a \neq c \neq b)
\end{cases}
\end{align*}
\]
Ground freshness theory

\[
\begin{align*}
(a \neq b) \\
\frac{a \not\# b}{a \not\# \langle \rangle} & \quad \text{Different names fresh} \\
\frac{a \not\# \langle \rangle}{a \not\# t} & \quad \text{Anything fresh for unit} \\
\frac{a \not\# t}{a \not\# f(t)} & \quad \text{Freshness ignores function symbols} \\
\frac{a \not\# t \quad a \not\# u}{a \not\# \langle t, u \rangle} & \quad \text{Freshness ignores pairs} \\
\frac{a \not\# \langle a \rangle t}{a \not\# \langle a \rangle t} & \quad \text{Fresh if bound} \\
\frac{(a \neq b) \quad a \not\# t}{a \not\# \langle b \rangle t} & \quad \text{Fresh if fresh for body}
\end{align*}
\]
Ground equational theory

\[
\begin{align*}
\frac{a \approx a}{\langle \rangle \approx \langle \rangle} \\
\frac{t_1 \approx t_2}{t_1 \approx u_1 \quad t_2 \approx u_2} \\
\frac{\langle t_1, t_2 \rangle \approx \langle u_1, u_2 \rangle}{t \approx u} \\
\frac{f(t) \approx f(u)}{t \approx u} \\
\frac{\langle a \rangle t \approx \langle a \rangle u}{(a \not\equiv b) \quad a \not\equiv u \quad t \approx (a \ b) \cdot u} \\
\frac{\langle a \rangle t \approx \langle b \rangle u}{\alpha\text{-equivalence for abstractions}}
\end{align*}
\]

Standard equational rules
A (nominal) Horn clause is a formula of the form

\[ A :\neg B_1, \ldots, B_n \]

where \( A, B_1, \ldots, B_n \) are atomic formulas.

We interpret such a clause as the nominal logic formula

\[ \forall \vec{\alpha} . \forall \vec{X}. B_1 \land \cdots \land B_n \supset A \]

where \( \vec{\alpha} = FN(A, \vec{B}) \) and \( \vec{X} = FV(A, \vec{B}) \).
Proof search

Proof search in $\alpha$Prolog is *depth-first backchaining* just like in Prolog, except:

1. Both variables and atoms (names) are *freshened* when resolving against a clause.

2. UPG’s nominal unification algorithm is used instead of ordinary syntactic unification.

3. In addition to substitutions, answers can contain freshness constraints.
Proof search: Correctness?

αProlog proof search is **sound** with respect to nominal logic:

answers found by αProlog are logical consequences of the corresponding theory

The big question: Is αProlog proof search **complete**?

can αProlog find all answers (at least in principle)?
No.
Counterexample

Program clauses:

\[ \forall a.p(a) \]

Goal:

\[ p(a) \]

Proof search fails because we freshen \( a \) in program clause \( p(a) \), so that the nominal unification step

\[ p(a') \approx p(a) \]

fails: logically equivalent but not equal nominal terms
The fly in the ointment
Problem: Equivariance

- In nominal logic, \textit{truth is preserved by name-swapping}

- Two atomic formulas (or rewrite rules) can be \textit{logically equivalent} but not \textit{equal} as nominal terms.

- Example:
  \[ p(a) \iff p((a \cdot b) \cdot a) \approx p(b) \quad \text{but} \quad p(a) \not\approx p(b) \]

- For complete proof search need to \textit{unify modulo equivariance}
Two reasonable reactions

• The hacker: Grr! Interesting problems! Must solve!
  
  – Unfortunately, full nominal and equivariant unification are \( \text{NP} \)-hard and algorithmically nontrivial. (I found this out the hard way.)

• The theorist: Bleh! Hard problems! Must avoid!
  
  – Unfortunately, some interesting programs require equivariance.
Why is this hard?

- Let’s take a little quiz.

- Satisfiable or not?
  \[ p((c \ b) \cdot X, X, (b \ a) \cdot Y, Y) \iff p(a, b, c, d) \]

- Satisfiable or not?
  \[ p((d \ c) \cdot X, X, (b \ a) \cdot Y, Y) \iff p(a, b, c, d) \]
Why is this hard?

• Let’s take a little quiz.

• Satisfiable or not?

\[ p((c \ b) \cdot X, X, (b \ a) \cdot Y, Y) \iff p(a, b, c, d) \]

No!

• Satisfiable or not?

\[ p((d \ c) \cdot X, X, (b \ a) \cdot Y, Y) \iff p(a, b, c, d) \]

Yes: \( X = c, Y = a, \) swap \((a \ d)(b \ c)\)

Nine cases to check
Another fun example

- Is this satisfiable?

\[ X \# (((X\ Y) \cdot (X\ Y) \cdot X\ (X\ Y) \cdot (X\ Y) \cdot X) \cdot X\ (X\ X) \cdot Y) \cdot Y \]
Another fun example

- Is this satisfiable? No

\[
X \# (((X Y) \cdot (X Y) \cdot X (X Y) \cdot (X Y) \cdot X) \cdot X (X X) \cdot Y) \cdot Y
\]
\[
\# ((X X) \cdot X (X X) \cdot Y) \cdot Y
\]
\[
\# (X Y) \cdot Y
\]
\[
\# X
\]
Avoiding equivariance
The idea

• Interpret equivariance *prescriptively*

• Everything will be fine as long as all the programs we write are *naturally equivariant*.

• Of course, checking this in general is undecidable (Rice’s Theorem).

• Plan: find syntactic restriction of clauses for which αProlog proof search is complete.
Obvious but doesn’t work

• *Obviously*, if the atomic formulas in our programs never have free names then we’re safe.

• Nope: program clause

\[ p(\langle a \rangle X, X). \]

has solution \( p(\langle a \rangle a, b) \) but \( \alpha \)Prolog doesn’t find this answer.

• Unsurprisingly, interaction between variables, names, and binding is subtle.
Short-cut

• Urban and I spent ages beating heads against walls on this so you don’t have to.

• A restricted nominal Horn clause is of the form

\[ \forall \vec{X}. A : \neg \exists \vec{Y}. B_1, \ldots, B_n \]

• RNHC’s are inherently equivariant (induction on derivations), so \( \alpha \)Prolog proof search is complete.

\[
\frac{\exists \vec{Y}. G(\vec{t})}{p(\vec{t})} \quad \frac{\exists \vec{Y}. G((b b') \cdot \vec{t})}{p((b b') \cdot \vec{t})}
\]
Examples

- The $\lambda$-typing rule can be rewritten as
  \[
  tc(G, \text{lam}(F), \text{arr}(T, U)) :\!-\! \forall a. F \approx \langle a \rangle E, tc(\langle (a, T) | G \rangle, E, U).
  \]
  This is equivalent (in spirit) to the original.

  So $tc$ is safe.

- On the other hand, $p(\langle a \rangle X, X)$ has no RNHC equivalent.

- Neither (without major surgery) does the second clause of $cconv$:
  \[
  cconv([y|G], \text{var}(x), E) = cconv(G, \text{var}(x), \pi 2(E)).
  \]
We need equivariant unification anyway.

- Urban and I developed a test for checking whether ordinary NHC’s are safe. It is based on equivariant unification.

- Also, evidently equivariant unification is required for some interesting programs anyway.

- Hacker: Grr!
Equivariant unification
Idea

- *Equivariant* unification: relax ground name restrictions of UPG, add permutation variables & inverses

\[
a, b, t, u ::= \langle \rangle \mid \langle t, u \rangle \mid f(t) \mid \langle a \rangle t \mid X \mid \Pi \cdot t \mid a
\]

\[
\Pi ::= (a \ b) \mid id \mid \Pi \circ \Pi' \mid \Pi^{-1} \mid P
\]

\[
C ::= t \approx u \mid a \neq t
\]

- \( t \) and \( u \) unify “up to a permutation” if \( P \cdot t \approx u \) is satisfiable.

- NP-hard [C 04]
Our approach

• Phase I: Get rid of term symbols (unit, pair, functions, abstractions)

• Phase II: Get rid of permutation operations (id, inverse, composition, swapping)

• This leaves problems of the form $P \cdot a \approx b$, $a \neq b$ only.

• Phase III: Solve remaining problems using permutation graphs
Our approach (I)

- First, get rid of unit, pair, function symbols and abstractions:

\[ a, b, t, u ::= \langle \rangle \mid \langle t, u \rangle \mid f(t) \mid \langle a \rangle t \mid X \mid \Pi \cdot t \mid a \]

\[ \Pi ::= (a \ b) \mid \text{id} \mid \Pi \circ \Pi' \mid \Pi^{-1} \mid P \]

\[ C ::= t \approx u \mid a \# t \]
Our approach (I)

- Reduction rules for equality in phase I:

  \[(\approx?_1)\]  \(S, \langle \rangle \approx? \langle \rangle \rightarrow_1 S\)

  \[(\approx?_x)\]  \(S, \langle t_1, t_2 \rangle \approx? \langle u_1, u_2 \rangle \rightarrow_1 S, t_1 \approx? u_1, t_2 \approx? u_2\)

  \[(\approx?f)\]  \(S, f(t) \approx? f(u) \rightarrow_1 S, t \approx? u\)

  \[(\approx?_{abs})\]  \(S, \langle a \rangle t \approx? \langle b \rangle u \rightarrow_1 \left\{ \begin{array}{l}
S, a \approx? b, t \approx? u \\
\lor S, a \#? u, t \approx? (a \ b) \cdot u
\end{array} \right\}\)

  \[(\approx?_{var})\]  \(S, \Pi \cdot X \approx? t \rightarrow_1 S[X := \Pi^{-1} \cdot t], X \approx? \Pi^{-1} \cdot t\)

(\text{where } X \not\in FV(t), X \in FV(S))

- Note the 2-way choice point in rule for abstraction

- Otherwise, rules similar to UPG algorithm
Our approach (I)

- Reduction rules for freshness in phase I:
  \[(\#?_1) \quad S, a \#? \langle \rangle \rightarrow_1 S\]
  \[(\#?_\times) \quad S, a \#? \langle u_1, u_2 \rangle \rightarrow_1 S, a \#? u_1, a \#? u_2\]
  \[(\#?_f) \quad S, a \#? f(u) \rightarrow_1 S, a \#? u\]
  \[(\#?_{abs}) \quad S, a \#? \langle b \rangle u \rightarrow_1 \begin{cases} S, a \approx? b \\ \lor S, a \#? u \end{cases}\]

- Note the 2-way choice point in rule for abstraction

- Otherwise, rules similar to UPG algorithm
Our approach (II)

- Next, get rid of complex permutation terms:

\[
\begin{align*}
  a, b, t, u & ::= \langle \rangle | \langle t, u \rangle | f(t) | \langle a \rangle t | X | \prod \cdot t | a \\
  \prod & ::= (a \ b) | \text{id} | \prod \circ \prod' | \prod^{-1} | P \\
  C & ::= t \approx u | a \ # \ t
\end{align*}
\]
Our approach (II)

- Reduction rules, phase II:

  \( (id) \quad S[\text{id} \cdot v] \rightarrow_2 S[v] \)

  \( (inv) \quad S[\Pi^{-1} \cdot v] \rightarrow_2 \exists X. S[X], \Pi \cdot X \approx v \)

  \( (comp) \quad S[\Pi \circ \Pi' \cdot v] \rightarrow_2 \exists X. S[\Pi \cdot X], \Pi' \cdot v \approx X \) \)

  \( (swap) \quad S[(a \ a') \cdot v] \rightarrow_2 \left\{ \begin{array}{l} S[a], a' \approx v \\ \lor S[a'], a \approx v \\ \lor \exists X. S[X], v \approx X, a \# X, a' \# X \end{array} \right\} \)

- Note the 3-way choice point in rule for swapping
Our approach (III)

• The remaining constraints involve only names, variables, and permutation variables.

\[ a, b, t, u ::= \langle \rangle | \langle t, u \rangle | f(t) | \langle a \rangle t | X | \Pi \cdot t | a \]
\[
\Pi ::= (a \; b) | id | \Pi \circ \Pi' | \Pi^{-1} | P
\]
\[
C ::= t \approx u | a \; \# \; t
\]

• Problems of this form can be solved by graph reduction in poly. time.

• Idea: Build a graph with “freshness”, and “permutation” edges; reduce using permutation laws
An example

- Here’s how to reduce a permutation graph corresponding to:

\[ QPPa \approx b \quad PQPa \approx b \quad PPa \approx b \quad PQP^{-1}a \# a \]
An example

- Here’s how to reduce a permutation graph corresponding to:

\[ QPPa \approx b \quad PQPa \approx b \quad PPa \approx b \quad PQP^{-1}a \neq a \]
An example

- Here’s how to reduce a permutation graph corresponding to:

\[ QPPa \approx b \quad PQPa \approx b \quad PPa \approx b \quad PQP^{-1}a \neq a \]
An example

Here’s how to reduce a permutation graph corresponding to:

\[ QPPa \approx b \quad PQPa \approx b \quad PPa \approx b \quad PQP^{-1}a \# a \]
An example

- Here’s how to reduce a permutation graph corresponding to:

\[ QPPa \approx b \quad PQPa \approx b \quad PPa \approx b \quad PQP^{-1}a \# a \]
An example

- Here’s how to reduce a permutation graph corresponding to:

\[ QPPa \approx b \quad PQPa \approx b \quad PPa \approx b \quad PQP^{-1}a \neq a \]
An example

- Here’s how to reduce a permutation graph corresponding to:

\[ QPPa \approx b \quad PQPa \approx b \quad PPa \approx b \quad PQP^{-1}a \not\approx a \]
An example

Here’s how to reduce a permutation graph corresponding to:

\[ QPPa \approx b \quad PQPa \approx b \quad PPa \approx b \quad PQP^{-1}a \neq a \]

Unsatisfiable because \( Qa \neq a \) and \( Qa \approx a \)
Results

• Phase I (term reduction): \textbf{NP} time, finitary (possible improvement to poly. time, unitary.)

• Phase II (permutation reduction): \textbf{NP} time, finitary

• Phase III (graph reduction): \textbf{P} time, unitary.

• Overall: \textbf{NP} time, finitely many answers.
Aside: Equivariant matching

- Recall that nondeterminism comes from abstractions and swappings only.

- Based on this observation, developed a PTIME case of equivariant matching

- Solves $P \cdot t \approx u$ when $t, u$ are “swapping-free”, that is, of the form

$$t, u ::= X \mid \langle \rangle \mid \langle t, u \rangle \mid f(t) \mid \langle a \rangle t \mid a$$

and $u$ is ground.
Future work

• Where do we go from here?

• The hacker: Grr! Time for some hacking!

• The theorist: Is there a better-behaved fragment of nominal logic? (e.g., programs with no name variables)
Conclusion

- Nominal logic: interesting, powerful, but tricky to automate.

- Nominal logic programming is a first step in this direction

- Future: Nominal logic in theorem proving? Nominal logical framework?

- Lots of interesting stuff to do!
Determinizing phase I

• Idea: Replace rules of the form

\[
\begin{align*}
\left( \approx_{abs} \right) & \quad S, \langle a \rangle t \approx? \langle b \rangle u \quad \rightarrow_1 \quad \left\{ \begin{array}{l}
S, a \approx? b, t \approx? u \\
\lor S, a \#? u, t \approx? (a \ b) \cdot u
\end{array} \right. \\
\left( \#_{abs} \right) & \quad S, a \#? \langle b \rangle u \quad \rightarrow_1 \quad \left\{ \begin{array}{l}
S, a \approx? b \\
\lor S, a \#? u
\end{array} \right.
\end{align*}
\]

• with deterministic rules

\[
\begin{align*}
\left( \approx_{abs} \right) & \quad \langle a \rangle t \approx? \langle b \rangle u \quad \rightarrow_1 \quad \mathcal{N}.(a \ c) \cdot t \approx? (b \ c) \cdot u \\
\left( \#_{abs} \right) & \quad a \#? \langle b \rangle u \quad \rightarrow_1 \quad \mathcal{N}.a \#? (b \ c) \cdot u
\end{align*}
\]

• Problem: more swappings so maybe more nondeterminism later