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!

αProlog

- αProlog is a logic programming language based on Pitts' nominal logic

- and using Urban, Pitts, and Gabbay's nominal unification algorithm

- FreshML : ML :: αProlog : 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.

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma \vdash e : T \to U \quad \Gamma \vdash f : T}{\Gamma \vdash e\ f : U} \qquad \frac{(x \notin \Gamma) \quad \Gamma, x : T \vdash e : U}{\Gamma \vdash \lambda x.e : T \to U}$$

```
tc(G,var(X),T)              :- mem((X,T),G).
tc(G,app(E,F),U)            :- tc(G,E,arr(T,U)), tc(G,F,T).
tc(G,lam(x\E),arr(T,U)) :- x # G, tc([(x,T)|G],E,U).
```

Examples (II)

- A less trivial example: big step semantics for λ_{ref}.

$$\frac{(a \in Lab)}{\langle M, a\rangle \rightarrow \langle M, a\rangle} \qquad \frac{\langle M, e_1\rangle \rightarrow \langle M', a\rangle \quad \langle M', e_2\rangle \rightarrow \langle M'', v\rangle}{\langle M, e_1 := e_2\rangle \rightarrow \langle M''[a := v], ()\rangle}$$

$$\frac{\langle M, e\rangle \rightarrow \langle M', a\rangle}{\langle M, !e\rangle \rightarrow \langle M', M'(a)\rangle} \qquad \frac{\langle M, e\rangle \rightarrow \langle M', v\rangle \quad (a \notin dom(M'))}{\langle M, \mathsf{ref}\ e\rangle \rightarrow \langle M'[a := v], a\rangle}$$

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

Examples (II)

- A less trivial example: big step semantics for λ_{ref}.

```
(M,lab(A))          `eval` (M,lab(A)).
(M,assign(E1,E2)) `eval` (M3,unit)            :- (M,E1) `eval` (M1,lab(A)),
                                                  (M1, E2) `eval` (M2,V),
                                                  update((A,V),M2,M3).
(M,deref(E))        `eval` (M',V)              :- (M,E) `eval` (M',lab(A)),
                                                  mem((A,V),M').
(M,ref(E))          `eval` ([(a,V)|M1],lab(a)) :- (M,E) `eval` ,(M1,V),
                                                  a#M1.
```

- Interesting part: last rule constrains И-quantified name to be sufficiently fresh

Examples (III)

- Another example: closure conversion

$$
\begin{array}{rcll}
C[\![x, \Gamma \vdash x]\!]e & = & \pi_1(e) & \\
C[\![y, \Gamma \vdash x]\!]e & = & C[\![\Gamma \vdash x]\!](\pi_2(e)) & (x \neq y) \\
C[\![\Gamma \vdash t_1 t_2]\!]e & = & \mathsf{let}\ c\ =\ C[\![\Gamma \vdash t_1]\!]e & \\
 & & \mathsf{in}\ (\pi_1(c))\ \langle C[\![\Gamma \vdash t_2]\!]e, \pi_2(c)\rangle & \\
C[\![\Gamma \vdash \lambda x.t]\!]e & = & \langle \lambda y.C[\![x, \Gamma \vdash t]\!]y, e\rangle & (x, y \notin \Gamma)
\end{array}
$$

Examples (III)

- Another example: closure conversion

```
cconv([x|G],var(x),E) = pi1(E).
cconv([y|G],var(x),E) = cconv(G,var(x),pi2(E)).
cconv(G,app(T1,T2),E)
        =  let(cconv(G,T1,E),c\
                app(pi1(var(c)),
                    pair(cconv(G,T2,E),pi2(var(c))))).
cconv(G,lam(x\T),E)
        =  pair(lam(y\cconv([x|G],T,var(y))),E)
        :- x#G,y#G.
```

- Note: Functions are unwound to relations in αProlog.

Nominal logic programming

Notation

a, b	$\in$	$\mathbb{A}$	Atoms/Names
f, g	$\in$	$FnSym$	Term symbols
X, Y	$\in$	Var	Variables
a, b, t, u	$::=$	$\langle\rangle \mid \langle t, u\rangle \mid f(t) \mid X$	First-order terms
	$\mid$	$\langle a\rangle t \mid \Pi \cdot t \mid \mathsf{a}$	Nominal terms
Π	$::=$	$(a\ b) \mid \mathsf{id} \mid \Pi \circ \Pi' \mid \Pi^{-1} \mid P$	Permutations
C	$::=$	$t \approx u \mid a \# t$	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 Π to a (ground) term is:

$$
\begin{array}{rcl}
\Pi \cdot \mathsf{a} & = & \Pi(\mathsf{a}) \\
\Pi \cdot \langle\rangle & = & \langle\rangle \\
\Pi \cdot \langle t, u\rangle & = & \langle \Pi \cdot t, \Pi \cdot u\rangle \\
\Pi \cdot f(t) & = & f(\Pi \cdot t) \\
\Pi \cdot \langle \mathsf{b}\rangle t & = & \langle \Pi \cdot \mathsf{b}\rangle \Pi \cdot t
\end{array}
$$

where

$$
\begin{array}{rcl}
\mathsf{id}(\mathsf{a}) & = & \mathsf{a} \\
\Pi \circ \Pi'(\mathsf{a}) & = & \Pi(\Pi'(\mathsf{a})) \\
(\mathsf{a}\ \mathsf{b})(\mathsf{c}) & = & \begin{cases} \mathsf{b} & (\mathsf{a} = \mathsf{c}) \\ \mathsf{a} & (\mathsf{b} = \mathsf{c}) \\ \mathsf{c} & (\mathsf{a} \neq \mathsf{c} \neq \mathsf{b}) \end{cases}
\end{array}
$$

Ground freshness theory

$$\frac{(\mathsf{a} \neq \mathsf{b})}{\mathsf{a} \# \mathsf{b}}$$ Different names fresh

$$\frac{}{\mathsf{a} \# \langle\rangle}$$ Anything fresh for unit

$$\frac{\mathsf{a} \# t}{\mathsf{a} \# f(t)}$$ Freshness ignores function symbols

$$\frac{\mathsf{a} \# t \quad \mathsf{a} \# u}{\mathsf{a} \# \langle t, u\rangle}$$ Freshness ignores pairs

$$\frac{}{\mathsf{a} \# \langle \mathsf{a}\rangle t}$$ Fresh if bound

$$\frac{(\mathsf{a} \neq \mathsf{b}) \quad \mathsf{a} \# t}{\mathsf{a} \# \langle \mathsf{b}\rangle t}$$ Fresh if fresh for body

Ground equational theory

$$\left.\begin{array}{c}
\overline{\mathsf{a} \approx \mathsf{a}} \\[1ex]
\overline{\langle\rangle \approx \langle\rangle} \\[1ex]
\dfrac{t_1 \approx u_1 \quad t_2 \approx u_2}{\langle t_1, t_2\rangle \approx \langle u_1, u_2\rangle} \\[1ex]
\dfrac{t \approx u}{f(t) \approx f(u)} \\[1ex]
\dfrac{t \approx u}{\langle \mathsf{a}\rangle t \approx \langle \mathsf{a}\rangle u}
\end{array}\right\}$$

Standard equational rules

$$\frac{(\mathsf{a} \neq \mathsf{b}) \quad \mathsf{a} \mathrel{\#} u \quad t \approx (\mathsf{a}\ \mathsf{b}) \cdot u}{\langle \mathsf{a}\rangle t \approx \langle \mathsf{b}\rangle u}$$

α-equivalence for abstractions

Nominal Horn clauses

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

$$A :- B_1, \ldots, B_n$$

where $A, B_1, \ldots, B_n$ are atomic formulas.

We interpret such a clause as the nominal logic formula

$$\text{И}\vec{a}.\forall \vec{X}.B_1 \wedge \cdots \wedge B_n \supset A$$

where $\vec{a} = FN(A, \vec{B})$ and $\vec{X} = FV(A, \vec{B})$.

Proof search

Proof search in αProlog 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:

$$\text{И}\mathsf{a}.p(\mathsf{a})$$

Goal:

$$p(\mathsf{a})$$

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

$$p(\mathsf{a}') \approx p(\mathsf{a})$$

fails: logically equivalent but not equal nominal terms

The fly in the ointment

Problem: Equivariance

- In nominal logic, *truth is preserved by name-swapping*

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

- Example:

$$p(\mathsf{a}) \iff p((\mathsf{a}\ \mathsf{b}) \cdot \mathsf{a}) \approx p(\mathsf{b}) \quad \text{but} \quad p(\mathsf{a}) \not\approx p(\mathsf{b})$$

- For complete proof search need to *unify modulo equivariance*

Two reasonable reactions

- The hacker: Grr! Interesting problems! Must solve!
  - Unfortunately, full nominal and equivariant unification are **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?

$$p((\mathsf{c}\ \mathsf{b}) \cdot X, X, (\mathsf{b}\ \mathsf{a}) \cdot Y, Y) \iff p(\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d})$$

- Satisfiable or not?

$$p((\mathsf{d}\ \mathsf{c}) \cdot X, X, (\mathsf{b}\ \mathsf{a}) \cdot Y, Y) \iff p(\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d})$$

Why is this hard?

- Let's take a little quiz.

- Satisfiable or not?

$$p((\mathsf{c}\ \mathsf{b}) \cdot X, X, (\mathsf{b}\ \mathsf{a}) \cdot Y, Y) \iff p(\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d})$$

  No!

- Satisfiable or not?

$$p((\mathsf{d}\ \mathsf{c}) \cdot X, X, (\mathsf{b}\ \mathsf{a}) \cdot Y, Y) \iff p(\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d})$$

  Yes: $X = \mathsf{c}, Y = \mathsf{a}$, 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

$$
\begin{aligned}
X \;\; &\# \;\; (((X\ Y) \cdot (X\ Y) \cdot X\ (X\ Y) \cdot (X\ Y) \cdot X) \cdot X\ (X\ X) \cdot Y) \cdot Y \\
&\# \;\; ((X\ X) \cdot X\ (X\ X) \cdot Y) \cdot Y \\
&\# \;\; (X\ Y) \cdot Y \\
&\# \;\; X
\end{aligned}
$$

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

$$p(\langle \mathsf{a} \rangle X, X).$$

  has solution $p(\langle \mathsf{a} \rangle \mathsf{a}, \mathsf{b})$ but αProlog 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

$$\forall \vec{X}.A :- \text{И}\vec{a}.\exists \vec{Y}.B_1, \ldots, B_n$$

- RNHC's are inherently equivariant (induction on derivations), so αProlog proof search is complete.

$$\frac{\text{И}\vec{a}.\exists \vec{Y}.G(\vec{t})}{p(\vec{t})} \implies \frac{\text{И}\vec{a}.\exists \vec{Y}.G((b\ b') \cdot \vec{t})}{p((b\ b') \cdot \vec{t})}$$

Examples

- The λ-typing rule can be rewritten as

  $$tc(G, lam(F), arr(T, U)) :- \text{И}\mathsf{a}.F \approx \langle \mathsf{a} \rangle E, tc([(\mathsf{a}, T)|G], E, U).$$

  This is equivalent (in spirit) to the original.

  So tc is safe.

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

- Neither (without major surgery) does the second clause of $cconv$:

  $$cconv([\mathsf{y}|G], var(\mathsf{x}), E) = cconv(G, var(\mathsf{x}), pi2(E)).$$

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

$$
\begin{aligned}
a, b, t, u &::= \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a\rangle t \mid X \mid \Pi \cdot t \mid \mathsf{a} \\
\Pi &::= (a\ b) \mid \mathsf{id} \mid \Pi \circ \Pi' \mid \Pi^{-1} \mid P \\
C &::= t \approx u \mid a \# t
\end{aligned}
$$

- t and u unify "up to a permutation" if $P \cdot t \approx u$ is satisfiable.

- $\mathbf{NP}$-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 $P \cdot a \approx b$, $a \# b$ only.

- Phase III: Solve remaining problems using *permutation graphs*

Our approach (I)

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

$$
\begin{array}{rcl}
a, b, t, u & ::= & \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a\rangle t \mid X \mid \mathsf{\Pi} \cdot t \mid \mathsf{a} \\
\mathsf{\Pi} & ::= & (a\ b) \mid \mathsf{id} \mid \mathsf{\Pi} \circ \mathsf{\Pi}' \mid \mathsf{\Pi}^{-1} \mid P \\
C & ::= & t \approx u \mid a \# t
\end{array}
$$

Our approach (I)

- Reduction rules for equality in phase I:

$$
\begin{array}{lrcl}
(\approx ?_1) & S, \langle\rangle \approx ? \langle\rangle & \to_1 & S \\
(\approx ?_\times) & S, \langle t_1, t_2 \rangle \approx ? \langle u_1, u_2 \rangle & \to_1 & S, t_1 \approx ? u_1, t_2 \approx ? u_2 \\
(\approx ?_f) & S, f(t) \approx ? f(u) & \to_1 & S, t \approx ? u \\
(\approx ?_{abs}) & S, \langle a \rangle t \approx ? \langle b \rangle u & \to_1 & \left\{ \begin{array}{c} S, a \approx ? b, t \approx ? u \\ \vee\, S, a \# ? u, t \approx ? (a\ b) \cdot u \end{array} \right\} \\
(\approx ?_{var}) & S, \Pi \cdot X \approx ? t & \to_1 & S[X := \Pi^{-1} \cdot t], X \approx ? \Pi^{-1} \cdot t \\
 & & & (\text{where } X \notin FV(t), X \in FV(S))
\end{array}
$$

- 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:

$$\begin{array}{lrcl}
(\#?_1) & S, a \ \#? \ \langle\rangle & \to_1 & S \\
(\#?_\times) & S, a \ \#? \ \langle u_1, u_2 \rangle & \to_1 & S, a \ \#? \ u_1, a \ \#? \ u_2 \\
(\#?_f) & S, a \ \#? \ f(u) & \to_1 & S, a \ \#? \ u \\
(\#?_{abs}) & S, a \ \#? \ \langle b \rangle u & \to_1 & \left\{ \begin{array}{c} S, a \approx? \ b \\ \vee \ S, a \ \#? \ u \end{array} \right\}
\end{array}$$

- 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:

$$
\begin{array}{rcl}
a, b, t, u & ::= & \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a\rangle t \mid X \mid \mathsf{\Pi} \cdot t \mid \mathsf{a} \\
\mathsf{\Pi} & ::= & (a\ b) \mid \mathsf{id} \mid \mathsf{\Pi} \circ \mathsf{\Pi}' \mid \mathsf{\Pi}^{-1} \mid P \\
C & ::= & t \approx u \mid a \# t
\end{array}
$$

Our approach (II)

- Reduction rules, phase II:

$$
\begin{array}{llll}
(id) & S[\mathsf{id} \cdot v] & \rightarrow_2 & S[v] \\
(inv) & S[\Pi^{-1} \cdot v] & \rightarrow_2 & \exists X. S[X], \Pi \cdot X \approx v \\
(comp) & S[\Pi \circ \Pi' \cdot v] & \rightarrow_2 & \exists X. S[\Pi \cdot X], \Pi' \cdot v \approx X) \\
(swap) & S[(a\ a') \cdot v] & \rightarrow_2 & \left\{ \begin{array}{c} S[a], a' \approx v \\ \vee\, S[a'], a \approx v \\ \vee\, \exists X. S[X], v \approx X, a \# X, a' \# X \end{array} \right\}
\end{array}
$$

- Note the 3-way choice point in rule for swapping

Our approach (III)

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

$$\begin{array}{rcl} a, b, t, u & ::= & \langle\rangle \mid \langle t, u\rangle \mid f(t) \mid \langle a\rangle t \mid X \mid \mathsf{\Pi} \cdot t \mid \mathsf{a} \\ \mathsf{\Pi} & ::= & (a\ b) \mid \mathsf{id} \mid \mathsf{\Pi} \circ \mathsf{\Pi}' \mid \mathsf{\Pi}^{-1} \mid P \\ C & ::= & t \approx u \mid a \# t \end{array}$$

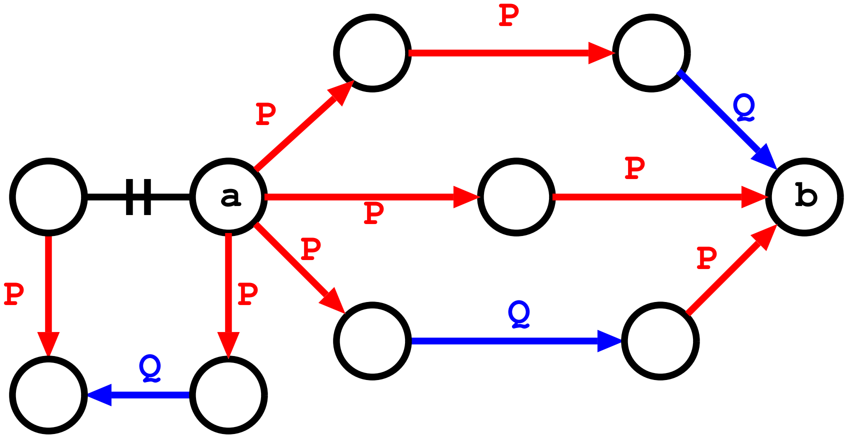
- Problems of this form can be solved by graph reduction in poly. time.

- 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:

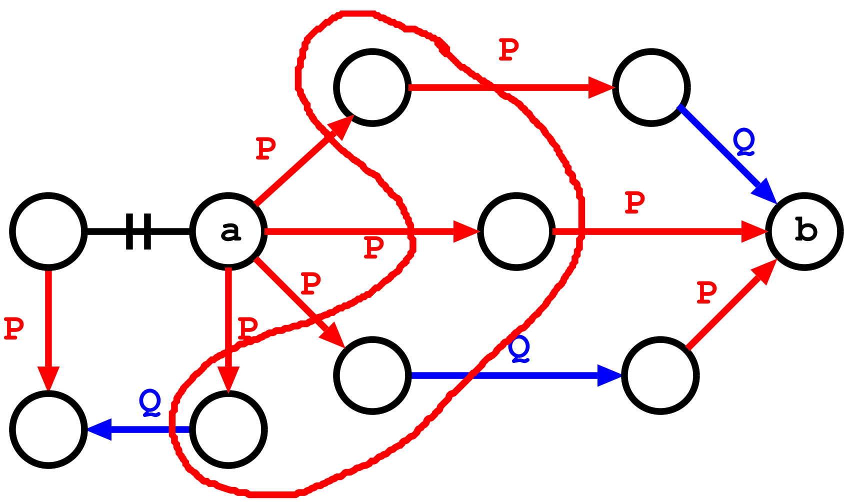
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

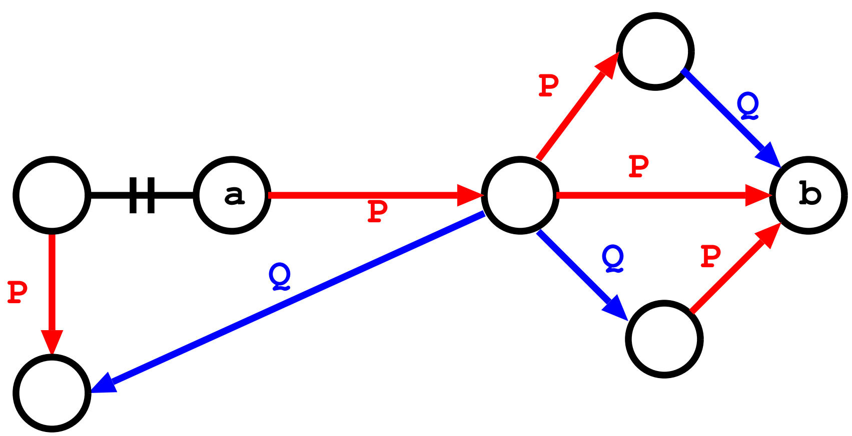
$$QP P\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

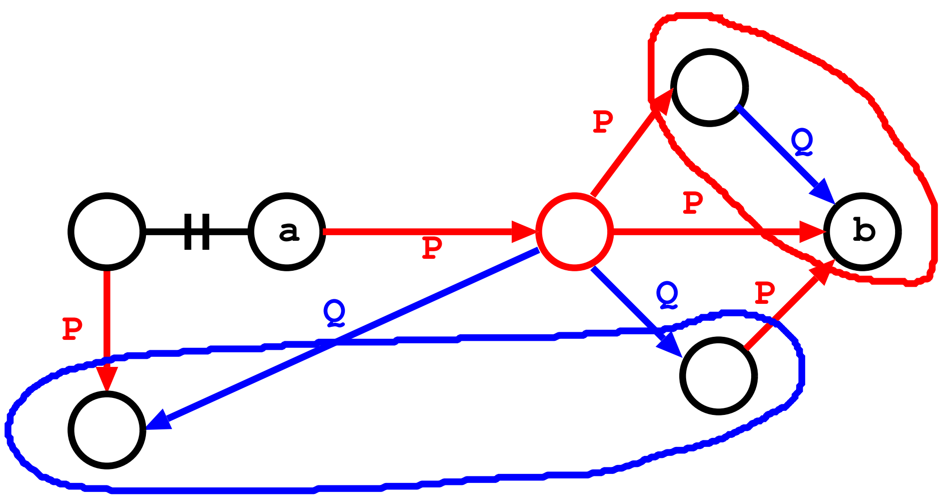
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

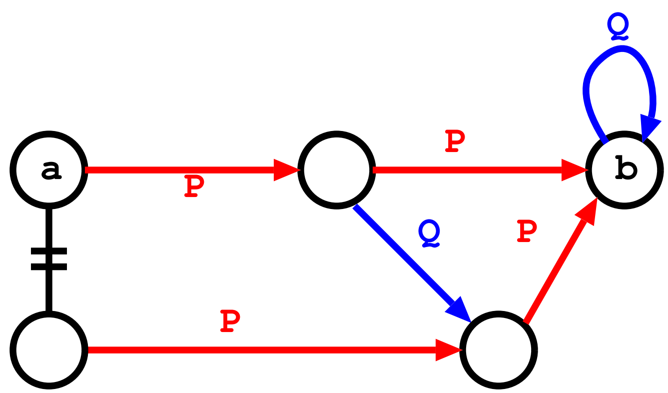
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

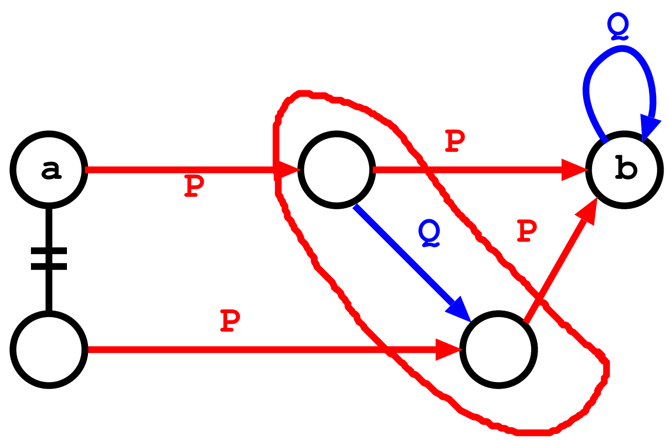
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

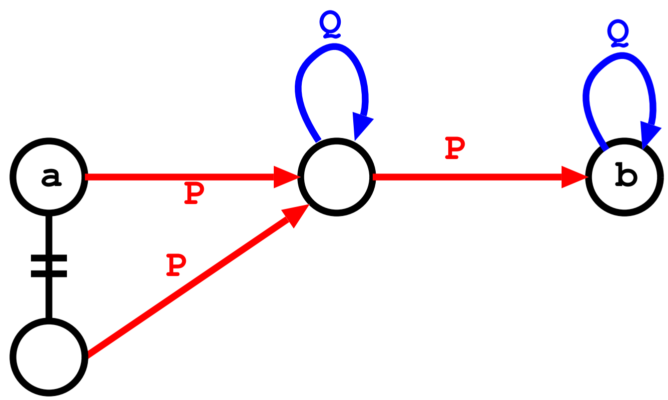
$$QP P\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

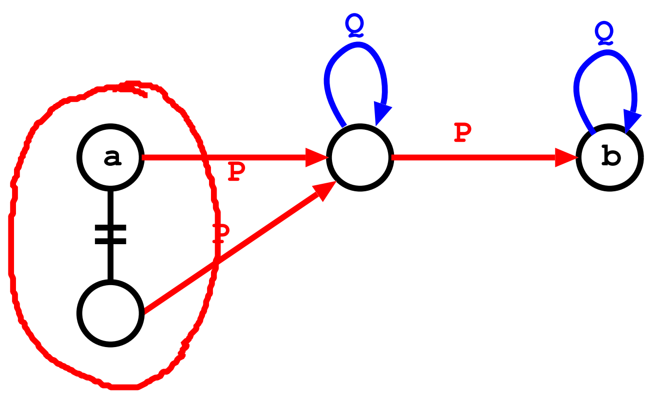
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

$$QP P\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



- Unsatisfiable because $Q\mathsf{a} \# \mathsf{a}$ and $Q\mathsf{a} \approx \mathsf{a}$

Results

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

- Phase $\mathbf{II}$ (permutation reduction): $\mathbf{NP}$ time, finitary

- Phase $\mathbf{III}$ (graph reduction): $\mathbf{P}$ time, unitary.

- Overall: $\mathbf{NP}$ 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

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

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

$$(\approx ?_{abs}) \quad S, \langle a\rangle t \approx ?\ \langle b\rangle u \ \to_1 \left\{ \begin{array}{c} S, a \approx ?\ b, t \approx ?\ u \\ \vee\ S, a \ \#?\ u, t \approx ?\ (a\ b) \cdot u \end{array} \right\}$$

$$(\#?_{abs}) \quad S, a\ \#?\ \langle b\rangle u \ \to_1 \left\{ \begin{array}{c} S, a \approx ?\ b \\ \vee\ S, a\ \#?\ u \end{array} \right\}$$

- with deterministic rules

$$\begin{array}{lrcl} (\approx ?_{abs}) & \langle a\rangle t \approx ?\ \langle b\rangle u & \to_1 & \text{И}\mathsf{c}.(a\ \mathsf{c}) \cdot t \approx ?\ (b\ \mathsf{c}) \cdot u \\ (\#?_{abs}) & a\ \#?\ \langle b\rangle u & \to_1 & \text{И}\mathsf{c}.a\ \#?\ (b\ \mathsf{c}) \cdot u \end{array}$$

- Problem: more swappings so maybe more nondeterminism later