Equivariant Unification

James Cheney

University of Edinburgh

RTA 2005

April 19, 2005

Motivation

- *Nominal logic* [Pitts 2003]: a variant of first-order logic with names and name-binding formalized using *swapping* (invertible renamings) and *freshness* ($- \notin FV(-)$).

- Goals: *term rewriting*, *automated/computer assisted reasoning*, and *logic programming* using nominal logic

- As for other theories, *unification* and *matching* are important decision procedures.

Motivation

- Previous work: Urban, Pitts, and Gabbay's *nominal unification* algorithm

- Nice properties: PTIME, unique most general unifiers

- Problem 1: Only handles a special case

- Problem 2: Equivariance: nominal resolution $\neq$ nominal unification

Notation

$$\begin{array}{rcll}
\mathsf{a}, \mathsf{b} & \in & \mathbb{A} & \text{Names} \\
f, g & \in & FnSym & \text{Uninterpreted function symbols} \\
X, Y & \in & Var & \text{Variables} \\
a, b, t, u & ::= & X \mid \langle\rangle \mid \langle t, u \rangle \mid f(t) & \text{First-order terms} \\
 & \mid & \langle a \rangle t \mid \Pi \cdot t \mid \mathsf{a} & \text{Nominal terms} \\
\Pi & ::= & (a\ b) \mid \mathsf{id} \mid \Pi \circ \Pi' \mid \Pi^{-1} \mid P & \text{Permutations} \\
C & ::= & t \approx u \mid a \# t & \text{Equality, freshness constraints}
\end{array}$$

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

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

α-equivalence for abstractions

Problem 1: UPG algorithm only solves a special case

- UPG algorithm does not consider problems involving *unknown names* in swappings or binding position

- This is why the algorithm remains unitary.

- For example, $(A\ B) \cdot C \approx C$ has two distinct solutions:

$$\{A = B\} \qquad \{A \# C, B \# C\}$$

Problem 2: 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 backchaining and rewriting, need to *unify/match modulo 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 $(a\ d)(b\ c)$

Wasn't that easy?

Another fun example

- Is this satisfiable?

$$X \quad \# \quad (((X\ Y) \cdot (X\ Y) \cdot X\ (X\ Y) \cdot (X\ Y) \cdot X) \cdot X\ (X\ X) \cdot Y) \cdot (X\ Y) \cdot$$

Another fun example

- Is this satisfiable? No

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

Outline

- UPG nominal unification

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

- Note: names a, b in $(\mathsf{a}\ \mathsf{b})$, $\langle \mathsf{a}\rangle t$, $\mathsf{a} \# t$ *must be ground*.

Outline

- Full nominal unification: allow name-variables anywhere.

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

- This is NP-complete because guessing is needed to deal with swapping [C 04]

Outline

- *Equivariant* unification: allow permutation variables & inverses

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

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

- Also 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:

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

Our approach (I)

- Reduction rules for equality in phase I:

$$
\begin{array}{llcl}
(\approx?_1) & S, \langle\rangle \approx? \langle\rangle & \rightarrow_1 & S \\
(\approx?_\times) & S, \langle t_1, t_2\rangle \approx? \langle u_1, u_2\rangle & \rightarrow_1 & S, t_1 \approx? u_1, t_2 \approx? u_2 \\
(\approx?_f) & S, f(t) \approx? f(u) & \rightarrow_1 & S, t \approx? u \\
(\approx?_{abs}) & S, \langle a\rangle t \approx? \langle b\rangle u & \rightarrow_1 & \left\{ \begin{array}{c} S, a \approx? b, t \approx? u \\ \vee\ S, a \#? u, t \approx? (a\ b) \cdot u \end{array} \right\} \\
(\approx?_{var}) & S, \Pi \cdot X \approx? t & \rightarrow_1 & S[X := \Pi^{-1} \cdot t], X \approx? \Pi^{-1} \cdot t \\
 & & & \text{(where } X \notin FV(t), X \in FV(S))
\end{array}
$$

- 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:

$$
\begin{array}{llcl}
(\#?_1) & S, a \; \#? \; \langle\rangle & \longrightarrow_1 & S \\
(\#?_\times) & S, a \; \#? \; \langle u_1, u_2 \rangle & \longrightarrow_1 & S, a \; \#? \; u_1, a \; \#? \; u_2 \\
(\#?_f) & S, a \; \#? \; f(u) & \longrightarrow_1 & S, a \; \#? \; u \\
(\#?_{abs}) & S, a \; \#? \; \langle b \rangle u & \longrightarrow_1 & \left\{ \begin{array}{c} S, a \approx ? \; b \\ \vee \; S, a \; \#? \; u \end{array} \right\}
\end{array}
$$

- 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:

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

Our approach (II)

- Reduction rules, phase II:

$$
\begin{array}{lrcl}
(id) & S[\mathsf{id} \cdot v] & \to_2 & S[v] \\
(inv) & S[\Pi^{-1} \cdot v] & \to_2 & \exists X.S[X], \Pi \cdot X \approx v \\
(comp) & S[\Pi \circ \Pi' \cdot v] & \to_2 & \exists X.S[\Pi \cdot X], \Pi' \cdot v \approx X) \\
(swap) & S[(a\ a') \cdot v] & \to_2 & \left\{ \begin{array}{c} S[a], a' \approx v \\ \vee\ S[a'], a \approx v \\ \vee\ \exists X.S[X], v \approx X, a \mathrel{\#} X, a' \mathrel{\#} X \end{array} \right\} \\
(\#_Q) & S, Q \cdot v \mathrel{\#} w & \to_2 & \exists X.S, Q \cdot v \approx X, X \mathrel{\#} w
\end{array}
$$

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

Our approach (III)

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

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

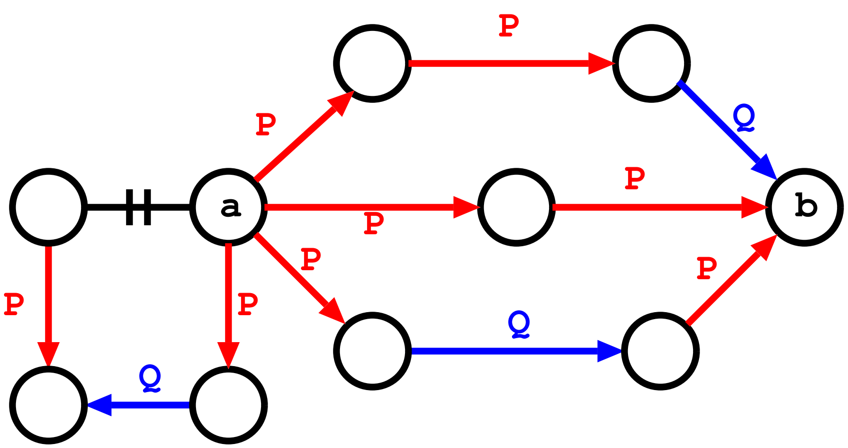
- Problems of this form can be solved by graph reduction in PTIME.

- Idea: Build a graph with “equality”, “freshness”, and “permutation” edges; reduce using permutation laws

An example

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

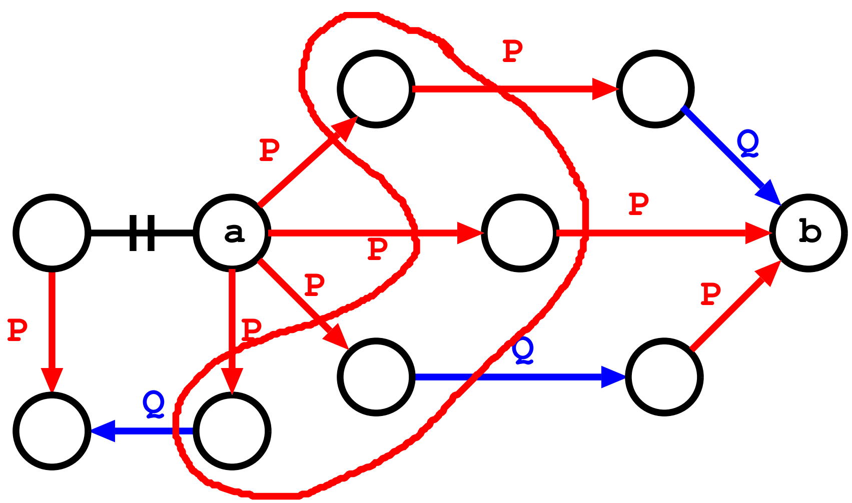
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

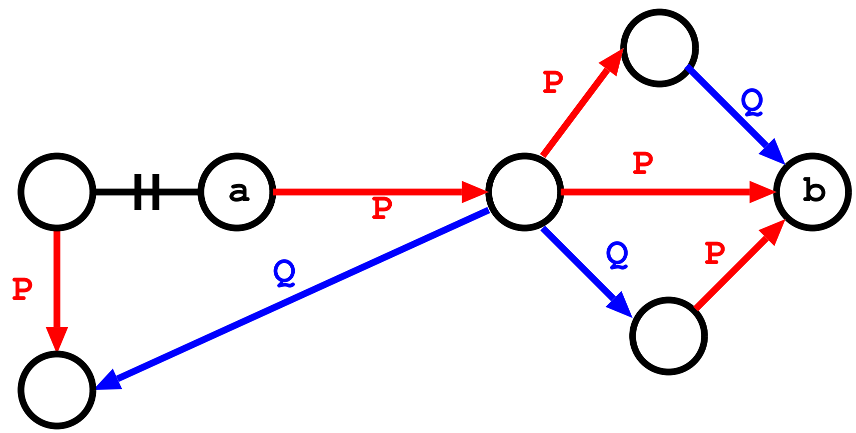
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

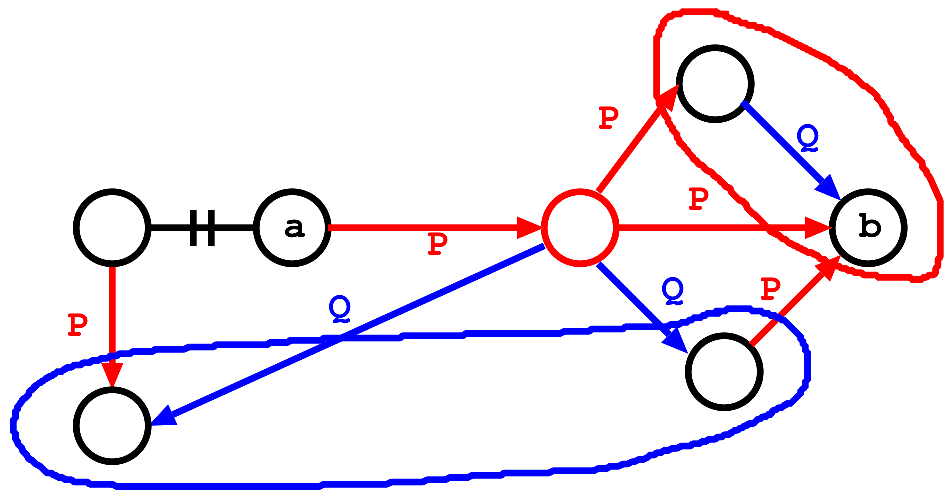
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

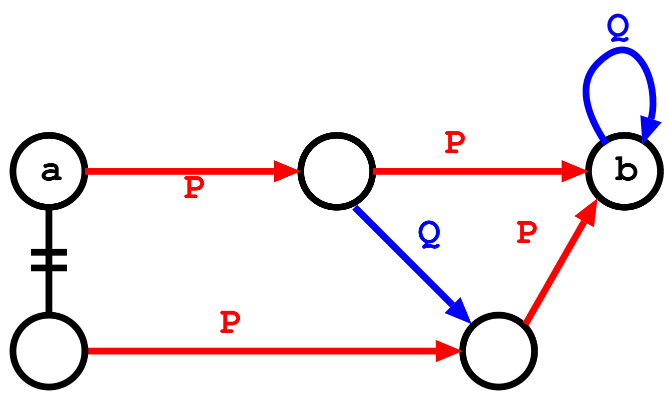
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



An example

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

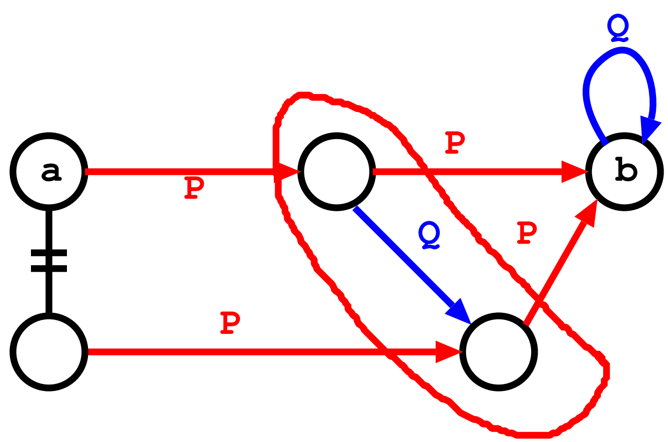
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \not\approx \mathsf{a}$$



An example

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

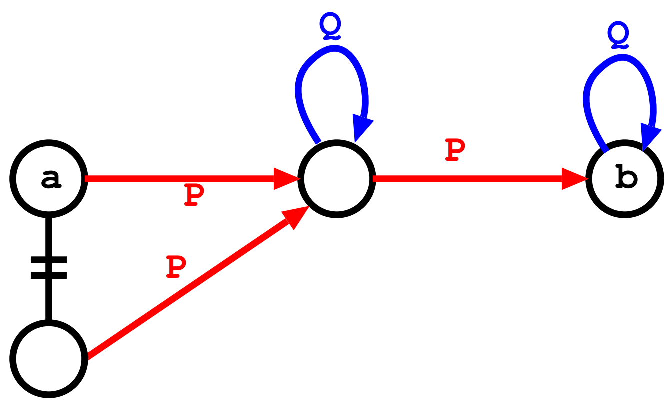
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \not\approx \mathsf{a}$$



An example

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

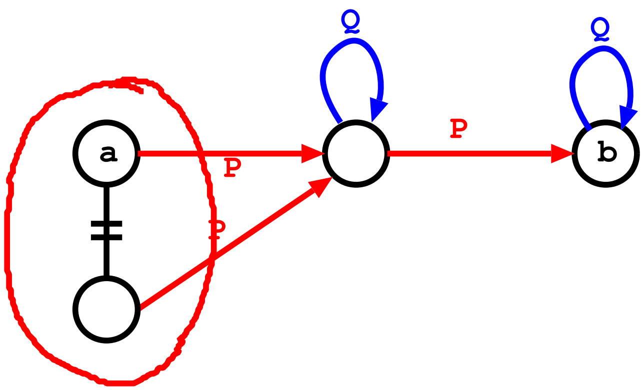
$$QPP\mathsf{a} \approx \mathsf{b} \qquad PQP\mathsf{a} \approx \mathsf{b} \qquad PP\mathsf{a} \approx \mathsf{b} \qquad PQP^{-1}\mathsf{a} \# \mathsf{a}$$



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 I (term reduction): NP time, finitary (possible improvement to PTIME, unitary.)
- Phase II (permutation reduction): NP time, finitary
- Phase III (graph reduction): P time, unitary.
- Overall: NP time, finitely many answers.

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.

Related work

- Solving and counting solutions to *group equations* well-studied, but not *group action equations*.

- Our results complement recent work on *avoiding equivariance* in nominal term rewriting/logic programming [Fernandez et al. 2004; Urban+C 2005]

- FreshML [Shinwell et al. 2003] pattern matching doesn't need equivariance & restricts patterns to keep matching efficient.

Future work

- Prototyped using constraint handling rules, "real" implementation pending.

- Managing nondeterminism (delaying/residuation)?

- Finding satisfactory efficient special cases?

- Applications to E-unification of *nominal equational theories* (e.g., π-calculus)?

Conclusions

- Equivariant unification ("unification up to a permutation") is a difficult and previously unstudied problem arising in automated reasoning for nominal logic

- We have developed the first complete, terminating algorithm.

- Not the end of the story: experience with practical issues and common cases needed.

Determinizing phase I

- Idea: Replace rules of the form

$$(\approx ?_{abs}) \quad S, \langle a \rangle t \approx ? \langle b \rangle u \;\rightarrow_1\; \left\{ \begin{array}{c} S, a \approx ? b, t \approx ? u \\ \vee\, S, a \,\#?\, u, t \approx ? (a\ b) \cdot u \end{array} \right\}$$

$$(\#?_{abs}) \quad S, a \,\#?\, \langle b \rangle u \;\rightarrow_1\; \left\{ \begin{array}{c} S, a \approx ? b \\ \vee\, S, a \,\#?\, u \end{array} \right\}$$

- with deterministic rules

$$\begin{array}{llll} (\approx ?_{abs}) & \langle a \rangle t \approx ? \langle b \rangle u & \rightarrow_1 & \mathsf{N}\mathsf{c}.(a\ \mathsf{c}) \cdot t \approx ? (b\ \mathsf{c}) \cdot u \\ (\#?_{abs}) & a \,\#?\, \langle b \rangle u & \rightarrow_1 & \mathsf{N}\mathsf{c}.a \,\#?\, (b\ \mathsf{c}) \cdot u \end{array}$$

- Problem: more swappings so maybe more nondeterminism later