The Complexity of Equivariant Unfiication

James Cheney, Cornell University

ICALP

July 15, 2004

Background

- Recently Gabbay and Pitts (LICS 1999) developed a new theory of names and binding.
- Names are an *abstract data type* with constants $a, b, \ldots$
- $(a\ b) \cdot x$: the result of swapping names a and b in x.
- $a \# x$: *freshness* predicate asserting "a is fresh for x"
- $\langle a \rangle x$: the *abstraction* of a over t, i.e. x with a bound.

Nominal Logic

- *Nominal Logic* (Pitts 2003): a first-order logic with names, swapping, freshness, abstraction
- Goal: Extend *automated reasoning* techniques to nominal setting
- αProlog (CU, ICLP '04): logic programming based on nominal logic, *nominal unification* (UPG, CSL '03)
- Related: *Nominal Rewriting* (FGM, PPDP '04)

αProlog Programming examples

- Typechecking

$$
\begin{array}{lll}
tc(G, var(X), T) & :- & mem((X, T), G). \\
tc(G, app(E, E'), T) & :- & tc(G, E, T' \to T), tc(G, E', T'). \\
tc(G, lam(\langle x \rangle E), T \to T') & :- & x \,\#\, G, tc([(x, T)|G], E, T').
\end{array}
$$

- α-inequivalence

$$
\begin{array}{lll}
neq(var(x), var(y)). & & \\
neq(abs(E1, E2), abs(F1, F2)) & :- & neq(E1, F1). \\
neq(abs(E1, E2), abs(F1, F2)) & :- & neq(E2, F2). \\
neq(lam(\langle x \rangle E), lam(\langle x \rangle F)) & :- & neq(E, F). \\
neq(var(\_), app(\_, \_)). & & \\
neq(var(\_), lam(\_)). \ldots. & &
\end{array}
$$

αProlog Programming examples

- Typechecking

$$
\begin{array}{lll}
tc(G, var(X), T) & :- & mem((X, T), G). \\
tc(G, app(E, E'), T) & :- & tc(G, E, T' \to T), tc(G, E', T'). \\
tc(G, lam(\langle x \rangle E), T \to T') & :- & x \# G, tc([(x, T)|G], E, T').
\end{array}
$$

- α-inequivalence

$$
\begin{array}{lll}
neq(var(x), var(y)). & & \\
neq(abs(E1, E2), abs(F1, F2)) & :- & neq(E1, F1). \\
neq(abs(E1, E2), abs(F1, F2)) & :- & neq(E2, F2). \\
neq(lam(\langle x \rangle E), lam(\langle x \rangle F)) & :- & neq(E, F). \\
neq(var(\_), app(\_, \_)). & & \\
neq(var(\_), lam(\_)). \ldots . & &
\end{array}
$$

Problem

- Question: Is nominal unification-based backchaining a *complete* resolution procedure? No.

- *Equivariance principle:* meaning preserved by name-swapping:

$$p(\vec{t}) \iff p((a\ b) \cdot \vec{t}) \iff p(\pi \cdot \vec{t})$$

- I.E., $neq(var(x), var(y))$ equivalent to $neq(var(x'), var(y'))$, any $x' \neq y'$

- But not $neq(var(x'), var(x'))$

Simple terms

- Terms $t \in \mathbb{T}$ using swapping, names $a \in \mathbb{A}$, name-valued variables $X \in \mathbb{V}$ only.

$$\begin{array}{rcl} t \in \mathbb{T} & ::= & X \mid (a\ b) \cdot t \mid a \\ \vec{t} \in \mathbb{T}^n & = & (t_1, \ldots, t_n) \end{array}$$

- *Valuation:* $\theta : \mathbb{V} \to \mathbb{A}$, lifted to $\mathbb{T} \to \mathbb{T}, \mathbb{T}^n \to \mathbb{T}^n$

- *Permutation:* $\pi : \mathbb{A} \to \mathbb{A}$, lifted to $\mathbb{T} \to \mathbb{T}, \mathbb{T}^n \to \mathbb{T}^n$; write $\pi \cdot t$

Equational theory

- A few simple axioms (all we will need)

$$
\begin{array}{rcll}
(a\ b)\cdot a & \approx & b & \\
(a\ b)\cdot b & \approx & a & \\
(a\ b)\cdot c & \approx & c & (a \neq c \neq b) \\
(t_1,\ldots,t_n) \approx (u_1,\ldots,u_n) & \iff & \bigwedge_i t_i \approx u_i &
\end{array}
$$

Equivariant unification

- For complete resolution in αProlog (also nominal rewriting), need *equivariant unification/matching*, or unification/matching "up to a permutation"

$$EVMatch = \{t \sim? \ u \mid \exists \theta, \pi. \ \pi \cdot \theta(t) \approx \theta(u)\}$$

$$EVSat = \{t \lesssim? \ u \mid \exists \theta, \pi. \ \pi \cdot \theta(t) \approx u\}$$

- Questions: How hard in theory? How to solve in practice?

- This talk: complexity (NP-hard) and a (simple) efficient special case

The separation problem

- Let $\mathbb{A}^{(n)}$ be set of n-tuples of *distinct* names.
- Separation: Is there a valuation such that $\theta(\vec{t})$ is a sequence of distinct names?

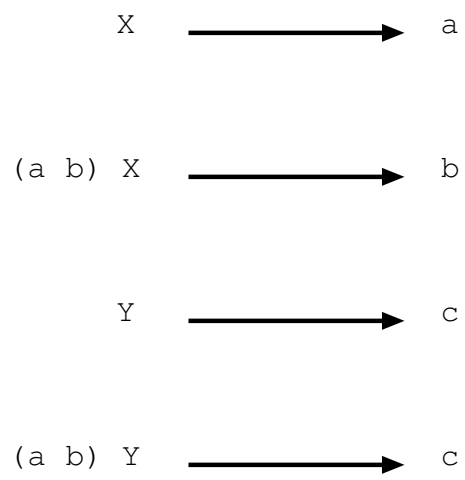
$$Sep = \{\vec{t} \in ?\ \mathbb{A}^{(n)} \mid \exists\theta.\ \theta(\vec{t}) \in \mathbb{A}^{(n)}\}$$

- Some easy reductions

$$Sep \leq_P EVMatch \leq_P EVSat$$

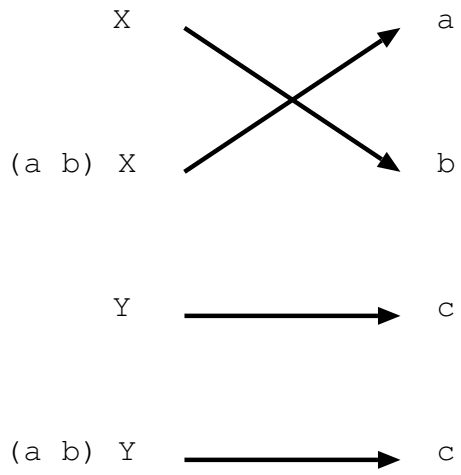
Examples

- An example



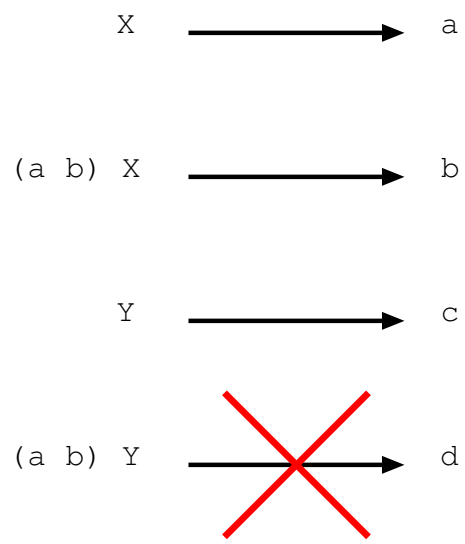
Examples

- An example, solution 2



Examples

- An unsatifsiable example, one case



Separation is NP-hard

- Graph 3-coloring reduces to separation.
- Graph over $\{1, \ldots, n\}$
- Idea: Vertices = variables $X_1, \ldots, X_n$.
- Colors = names $C = \{r, g, b\} \subset \mathbb{A}$.
- Coloring = valuation θ

Correctness (I)

- Need to constrain all $X_i \in C = \{r, g, b\}$.
- First attempt:

$$X \in C \iff X \neq (r\ g\ b) \cdot X \iff (X, (r\ g\ b) \cdot X) \in \mathbb{A}^{(2)}$$

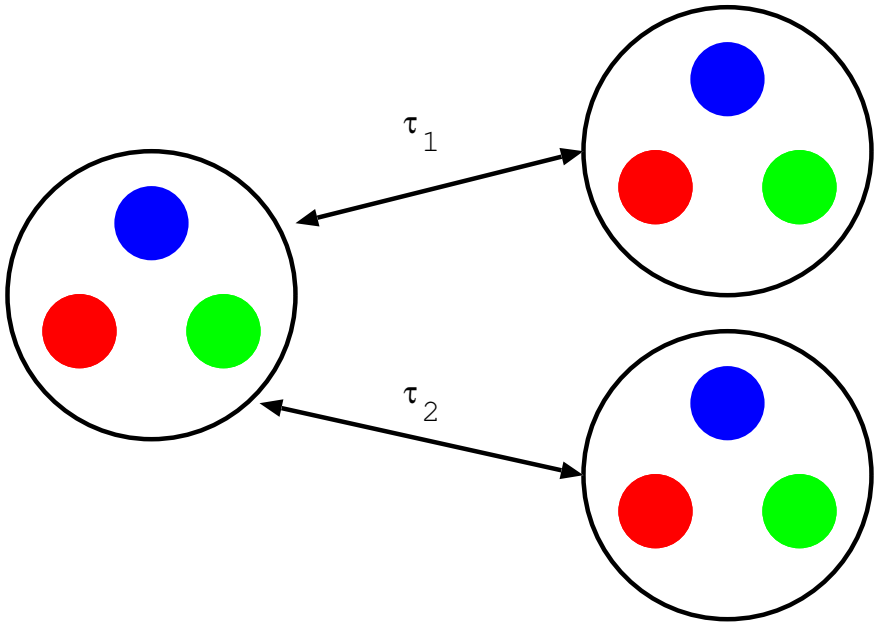
- Doesn't quite work:

$$X, Y \in C \not\Rightarrow (X, (r\ g\ b) \cdot X, Y, (r\ g\ b) \cdot Y) \in \mathbb{A}^{(4)}$$

for example, if $X = Y = r$.

Idea

- Use additional fresh r, g, b-colors, and permutations τ_i



...

Correctness (II)

- Use "fresh" color sets and permutations

$$\tau_1 = (r\ r_1)(g\ g_1)(b\ b_1), \tau_2 = (r\ r_2)(g\ g_2)(b\ b_2), \ldots$$

- to "hide" (or separate) the constraints from each other.

- Then

$$X_1, \ldots, X_n \in C \iff (\tau_1 \cdot X_1, \tau_1 \circ (r\ g\ b) \cdot X_1, \ldots) \in \mathbb{A}^{(2n)}$$

Correctness (III)

- Proper coloring: If (i, j) an edge, then $X_i \neq X_j$.

- For one edge, this works:

$$X_i \neq X_j \iff (X_i, X_j) \in \mathbb{A}^{(2)}$$

- Need to separate edge constraints with more new permutations σ:

$$\vec{t} = (\sigma_1 \cdot X_{e_1^s}, \sigma_1 \cdot X_{e_1^t}, \ldots) \in \mathbb{A}^{(2m)}$$

Correctness (IV)

- Need to verify that resulting problem

$$\vec{t} \in \mathbb{A}^{(2n+2m)}$$

  is equivalent to 3-coloring problem for G.

- Routine, but lots of cases.

  Theorem 1. *Sep $\leq_P$ 3-Graph Coloring*

Big picture

- Equivariant unification and matching also NP-hard.
- Reduction uses lots of swappings $\pi \cdot X$
- But explicit swappings are unusual in programs
- Efficient special cases?

First-order programs: a common case

- Typechecking

$$
\begin{array}{lll}
tc(G, var(X), T) & :- & mem((X, T), G). \\
tc(G, app(E, E'), T) & :- & tc(G, E, T' \to T)), tc(G, E', T'). \\
tc(G, lam(\langle x \rangle E), T \to T') & :- & x \# G, tc([(x, T)|G], E, T').
\end{array}
$$

- α-inequivalence

$$
\begin{array}{lll}
neq(var(x), var(y)). & & \\
neq(abs(E1, E2), abs(F1, F2)) & :- & neq(E1, F1); neq(E2, F2). \\
neq(lam(\langle x \rangle E), lam(\langle x \rangle F)) & :- & neq(E, F). \\
neq(var(\_), app(\_, \_)). & & \\
neq(var(\_), lam(\_)). \ldots . & &
\end{array}
$$

An efficient special case

- (*) Suppose t, u share no names or variables, and t has no abstractions, names, or swappings (i.e., t is *first-order*)

- Then

  Theorem 2. *If (*), then* $\exists\theta, \pi.\pi \cdot \theta(t) \approx \theta(u)$ *iff* $\exists\theta.\theta(t) \approx \theta(u)$

- That is, if (*) holds, then nominal unification = equivariant unification

Idea of the proof

- Let $t(\vec{X}), u(\vec{Y})$, θ, π be given with $\pi \cdot \theta(t) \approx \theta(u)$.
- Let

$$\theta' = \begin{cases} \pi \cdot \theta(X) & X \in FV(t) \\ \theta(Y) & Y \in FV(u) \end{cases}$$

- Then

$$\pi \cdot \theta(t) \approx u \iff \theta'(t) \approx \theta'(u)$$

- Other direction trivial.

Implications

- If $A :- G$ is a Horn clause and A is first-order, then $A \sim? A'$ iff $A \approx? A'$.
- If $t \to u$ is a first-order rewrite rule then $t \sim? t'$iff $t \approx? t'$
- First-order logic programs and rewrite rules can be run efficiently on nominal terms
- Expensive equivariance only needed when nominal features really used.

Summary

- Nominal logic programming (αProlog) provides advanced facilities for programming with names and binding
- NP-hardness of EV unification: a significant technical challenge
- But, efficient special cases exist, so high complexity can sometimes be avoided.
- Future/current work: Algorithms that work well in practice, additional efficient cases