Logic Mentoring Workshop - FLoC
Rewriting systems and applications

Maribel Fernández

July 2018

Motivations

Rewrite rules are an abstract way to specify transformations.

Ideal tool to model the dynamics of a system, analyse it, check properties.

Examples:

$$HCl + NaOH \to NaCl + H_20$$

$$\begin{array}{c} n + 0 \to n \\ n + S(m) \to S(n+m) \end{array}$$

$$\begin{array}{rcl} a\#P & \vdash & P \wedge \forall[a]Q \to \forall[a](P \wedge Q) \\ a\#P & \vdash & P \vee \exists[a]Q \to \exists[a](P \vee Q) \\ & \vdash & \neg(\exists[a]Q) \to \forall[a]\neg Q \\ & \vdash & \neg(\forall[a]Q) \to \exists[a]\neg Q \end{array}$$

Motivations

Rewriting systems as a modelling tool.

Expressivity:

- Rewriting is used to specify, in a uniform way, several computation paradigms (functional, logic, imperative and concurrent).

Motivations

Rewriting systems as a modelling tool.

Expressivity:

- Rewriting is used to specify, in a uniform way, several computation paradigms (functional, logic, imperative and concurrent).
- Rewriting is used to specify security features:
  - access control policies in various access control models (ACL, RBAC, CBAC), specified in a uniform and formal way.
  - analysis of cryptographic protocols

Motivations

Rewriting systems as a modelling tool.

Expressivity:

- Rewriting is used to specify, in a uniform way, several computation paradigms (functional, logic, imperative and concurrent).
- Rewriting is used to specify security features:
  - access control policies in various access control models (ACL, RBAC, CBAC), specified in a uniform and formal way.
  - analysis of cryptographic protocols
- Graph rewriting is used to specify complex systems in many areas: social networks, biochemical systems, interaction networks, software systems, etc.

Motivations

Why use rewriting systems as a modelling tool?

- A well-developed theory: rewriting techniques can be used to prove properties of the systems modelled (e.g., termination of a program, consistency of an access control policy, determinism, etc).

Motivations

Why use rewriting systems as a modelling tool?

- A well-developed theory: rewriting techniques can be used to prove properties of the systems modelled (e.g., termination of a program, consistency of an access control policy, determinism, etc).
- Availability of tools to test and experiment with evaluation strategies, to automate equational reasoning, and also for rapid prototyping: Maude, CiME, PORGY, GP, Kappa, ...

There are several kinds of rewriting systems in the literature.
My work has focused on:

- Term Rewriting (first and higher-order, and the "intermediate" **Nominal Rewriting** [FernandezGabbay2007])

There are several kinds of rewriting systems in the literature.
My work has focused on:

- Term Rewriting (first and higher-order, and the "intermediate" **Nominal Rewriting** [FernandezGabbay2007])
- The Lambda Calculus (paradigmatic model of functional computation)

Term Rewriting, Higher-order Rewriting, Graph Rewriting

There are several kinds of rewriting systems in the literature.
My work has focused on:

- Term Rewriting (first and higher-order, and the "intermediate" **Nominal Rewriting** [FernandezGabbay2007])
- The Lambda Calculus (paradigmatic model of functional computation)
- Graph Rewriting (interaction nets, portgraph rewriting cf. PORGY [CiE 2014])

Term Rewriting, Higher-order Rewriting, Graph Rewriting

There are several kinds of rewriting systems in the literature.
My work has focused on:

- Term Rewriting (first and higher-order, and the "intermediate" **Nominal Rewriting** [FernandezGabbay2007])
- The Lambda Calculus (paradigmatic model of functional computation)
- Graph Rewriting (interaction nets, portgraph rewriting cf. PORGY [CiE 2014])
- Development of rewrite-based modelling languages, including **strategy languages**, joint work with the PORGY team.

Applications

- Programming languages (semantics, typing):
  Nominal rewriting: completion algorithms, type systems (polymorphism, dependent type systems – logical frameworks)
- Computation models, proof systems, resource analysis (linearity)
- Access control systems:
  Policy composition (modularity properties of rewriting systems), type systems to check consistency and totality
- Software systems (Theory of specifications, model transformations): Domain-Specific versions of PORGY

Nominal Approach [Pitts, Gabbay]

Inspired by nominal set theory (Fraenkel-Mostowski).
Key ideas: Freshness conditions $a\#t$, name swapping $(a\ b)\cdot t$.
Example: β and η rules

$$\begin{array}{rlcl} & app(lam([a]Z), Z') & \rightarrow & subst([a]Z, Z') \\ a\#M \vdash & (\lambda([a]app(M, a)) & \rightarrow & M \end{array}$$

- Terms with binders
- Built-in α-equivalence
- Efficient matching and unification algorithms
- Simple notion of substitution ("first-order"), non-capturing substitution has to be specified using rewrite rules.
- Dependencies of terms on names are implicit.

Nominal Syntax [Urban,Pitts,Gabbay: TCS'04]

- Function symbols: $f, g \ldots$
  Variables: $M, N, X, Y, \ldots$
  Atoms: $a, b, \ldots$
  Swappings: $(a\ b)$
  Def. $(a\ b)a = b$, $(a\ b)b = a$, $(a\ b)c = c$
  Permutations: lists of swappings, denoted π (Id empty).
- Nominal Terms:

$$s, t ::= a \mid \pi \cdot X \mid [a]t \mid f\ t \mid (t_1, \ldots, t_n)$$

  $Id \cdot X$ written as X.
- Example (ML): $var(a)$, $app(t, t')$, $lam([a]t)$, $let(t, [a]t')$, $letrec[f]([a]t, t')$, $subst([a]t, t')$
  Syntactic sugar:
  a, (tt'), $\lambda a.t$, let $a = t$ in t', letrec $fa = t$ in t', $t[a \mapsto t']$

α-equivalence

$a\#X$ means $a \notin \mathrm{fv}(X)$ when X is instantiated (avoiding name capture).

$$\frac{}{a \approx_\alpha a} \qquad \frac{ds(\pi,\pi')\#X}{\pi \cdot X \approx_\alpha \pi' \cdot X}$$

$$\frac{s_1 \approx_\alpha t_1 \quad \cdots \quad s_n \approx_\alpha t_n}{(s_1,\ldots,s_n) \approx_\alpha (t_1,\ldots,t_n)} \qquad \frac{s \approx_\alpha t}{fs \approx_\alpha ft}$$

$$\frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \qquad \frac{a\#t \qquad s \approx_\alpha (a\ b)\cdot t}{[a]s \approx_\alpha [b]t}$$

where

$$ds(\pi,\pi') = \{n | \pi(n) \neq \pi'(n)\}$$

- $a\#X, b\#X \vdash (a\ b)\cdot X \approx_\alpha X$

α-equivalence

$a\#X$ means $a \notin \mathrm{fv}(X)$ when X is instantiated (avoiding name capture).

$$\frac{}{a \approx_\alpha a} \qquad \frac{ds(\pi,\pi')\#X}{\pi \cdot X \approx_\alpha \pi' \cdot X}$$

$$\frac{s_1 \approx_\alpha t_1 \quad \cdots \quad s_n \approx_\alpha t_n}{(s_1,\ldots,s_n) \approx_\alpha (t_1,\ldots,t_n)} \qquad \frac{s \approx_\alpha t}{fs \approx_\alpha ft}$$

$$\frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \qquad \frac{a\#t \qquad s \approx_\alpha (a\ b)\cdot t}{[a]s \approx_\alpha [b]t}$$

where

$$ds(\pi,\pi') = \{n | \pi(n) \neq \pi'(n)\}$$

- $a\#X, b\#X \vdash (a\ b) \cdot X \approx_\alpha X$
- $b\#X \vdash \lambda[a]X \approx_\alpha \lambda[b](a\ b) \cdot X$

Freshness

Also defined by induction:

$$\frac{}{a\#b} \qquad \frac{}{a\#[a]s} \qquad \frac{\pi^{-1}(a)\#X}{a\ \#\ \pi \cdot X}$$

$$\frac{a\#s_1 \quad \cdots \quad a\#s_n}{a\#(s_1, \ldots, s_n)} \qquad \frac{a\#s}{a\#fs} \qquad \frac{a\#s}{a\#[b]s}$$

Nominal Rewriting

Rewriting on nominal terms.

Applications:

- equational reasoning on data structures with binding;
- algebraic specifications of operators and data structures;
- operational semantics of programs;
- compilers, program transformations, etc.

Nominal Rewriting Rules:

$$\Delta \vdash l \to r \qquad V(r) \cup V(\Delta) \subseteq V(l)$$

Example: prenex normal forms in first-order logic

$$\begin{array}{lcl}
a\#P & \vdash & P \wedge \forall[a]Q \to \forall[a](P \wedge Q) \\
a\#P & \vdash & (\forall[a]Q) \wedge P \to \forall[a](Q \wedge P) \\
a\#P & \vdash & P \wedge \exists[a]Q \to \exists[a](P \wedge Q) \\
a\#P & \vdash & (\exists[a]Q) \wedge P \to \exists[a](Q \wedge P) \\
 & \vdash & \neg(\exists[a]Q) \to \forall[a]\neg Q \\
 & \vdash & \neg(\forall[a]Q) \to \exists[a]\neg Q
\end{array}$$

Reduction relation generated by (equivariant) nominal matching.

Nominal matching problem

$l \approx_\alpha t$ where $V(l) \cap V(t) = \emptyset$ has solution (Δ, θ) if

$$\Delta \vdash l\theta \approx_\alpha t$$

- Nominal matching is decidable [Urban, Pitts, Gabbay 2003]
- A solvable problem Pr has a unique most general solution: (Γ, θ) such that $\Gamma \vdash Pr\theta$.
- Efficient algorithms: linear in time and space [Calves-F.2008]
- If rules are **closed**, nominal matching is sufficient (otherwise, equivariant nominal matching — NP [Cheney2004]).

Termination

Many techniques for first-order systems: well developed area.

Example: recursive path ordering [Dershowitz]

rpo: A well-founded precedence on function symbols generates a well-founded ordering on first-order terms.
Nominal rpo: [ICALP2012]

Equational reasoning

If an equational theory can be represented by a confluent and terminating rewrite system, then equational reasoning can be mechanised.

*Closed rewriting is sufficient to decide equality modulo a nominal equational theory*if the axioms can be oriented to form a confluent and terminating closed NRS [LFMTP2010].
Completion procedure [ICALP 2012]

To solve equations: *nominal narrowing* [FSCD2016] using nominal unification instead of matching.

Equational reasoning: AC symbols

AC nominal equality: AC matching and unification
[LSFA 2016, LOPSTR 2017]

Commutativity: *infinitary unification theory* if unifiers are represented with freshness contexts and substitutions,
but *finitary* if using fixed point constraints and substitutions
[FSCD2018]

- Many sorted
- Polymorphic (Church/Curry styles) [TOCL 2018]
- Dependent types [TLCA 2015
- Intersection types [TCS 2018]

Future Work

- Nominal Terms: extension of first-order syntax, efficient matching modulo α.
- Clean semantics: Nominal Sets
- **Equational reasoning and rewriting**
  Completion as a tool to mechanise nominal equational logic
- Future work: functional abstraction and capture-avoiding substitution, implementing type checking algorithms, efficient AC-matching ...

Some (mentoring) conclusions:

Not really opposed:

- Theory vs Practice?
- Depth vs Breadth?
- Teaching vs Research?
- Diversity
- Positive approach... and team work