Graph representations of binders

Ian Mackie

ICMS Workshop on Mathematical Theories of Abstraction, Substitution and Naming in Computer Science
26-28 May 2007
Plan

- Binders
- Graphs
- Interaction nets
- Closed reduction
- Rewriting calculus
- Geometry of Interaction
- Conclusions/further work
The notion of binder is ubiquitous:

- programs
- programming language semantics
- module systems
- specifications of security features such as access control mechanisms
- computation models such as the $\lambda$-calculus, the $\pi$-calculus, etc.

These are examples of systems that are specified using reduction (rewrite) rules that involve binders.
Reasoning about binders

• To specify, understand, and reason about these systems, we need to have a good formalism where the notion of binding can be naturally accommodated.

• There are several tools to specify the dynamics of systems with binders. For instance: the rewriting calculus, higher-order rewrite systems. etc

These are all textual tools

• Graphical formalisms have clear advantages as modelling tools, in particular in the earlier phases of the specification.

• Graphical formalisms are more intuitive and make it easier to visualise the system (with or without binders). Examples: sequent calculus vs. proof nets, entity-relationship diagrams vs tables, etc.
Graphs

- Positive: problems that require a lot of attention in a textual notation disappear completely in a graphical formalism (see later)
- Negative: $G \rightarrow G'$
  implementation issues: pattern-matching is not an easy problem and can be very inefficient in general.

Interaction nets are a very specific form of graph rewriting that keep some of the positive aspects, but are well adapted to efficient implementation.

Use interaction nets when you want to implement graph rewriting
A universal model of computation, based on net rewriting

Set of agents:

\[
\alpha \quad \vdots \quad x_1 \quad \cdots \quad x_n
\]

Set of rewrite rules:

\[
\alpha \quad \rightarrow \quad \beta \quad \vdots \quad y_1 \quad \cdots \quad y_m
\]

\[
N \quad \vdash \quad \alpha \quad \rightarrow \quad \beta
\]

\[
\alpha \quad \vdots \quad x_1 \quad \cdots \quad x_n \quad \vdots \quad y_1 \quad \cdots \quad y_m
\]

At most one rule for each pair of agents; interface is preserved
Why Interaction Nets?

- Proving to be the leading formalism to talk about notions of “work”, sharing, and atomic computation steps. I.e. can be used as a cost model of computation.
- Language between specification and implementation
- Offers a low level operational semantics of computation
- Implementation technique (through compilation of other languages)
- Sequential and parallel implementations
• Able to represent abstraction, scope and binding
  Nothing specific in the graphs needs to be added
• Cannot simulate $\beta$-reduction in full generality: we need to fix a reduction strategy
• Closed reduction - one of the simplest yet most efficient strategies for the $\lambda$-calculus that I am aware of.
Graph representations of binders
Drawn like the abstract syntax tree, but need not be
Graph representations of binders
Closed reduction: interaction rules

\[ \lambda c \Rightarrow \lambda c \]

\[ b \Rightarrow \lambda c \]

\[ c \Rightarrow \lambda c \]

\[ \delta \Rightarrow \delta \]

\[ \nu \Rightarrow \lambda \]

\[ \lambda c \Rightarrow \lambda c \]
Properties: Substitution Lemma

\[(t[u/x])[v/y] = (t[v/y])[u/x]\]

\[(t[u/x])[v/y] = t[u[v/y]/x]\]

Note: where are the variable constraints in the diagrams?
• Many syntactic properties become identities: textual syntax is often too verbose. Other examples:

\[ x[v/x] = v \]
\[ (tu)[v/x] = (t[v/x])u \]
\[ (tu)[v/x] = t(u[v/x]) \]

• Interaction nets are confluent (by construction). We only need to show that there exits one sequence of reductions to the answer, then we know all reductions reach that same answer.
• Ideas can be extended to term rewriting systems with binders
• We show the case for the rewriting calculus (Kirchner et al.)
• Combines term rewriting with λ-calculus in a very natural way
• Interaction net understanding
• Apply other techniques that we know work well for λ-graph rewriting systems: Geometry of Interaction
\( \rho \)-calculus

Syntax:

\[
t, u ::= x \mid f \mid p \to t \mid [p \ll u].t \mid t \ u
\]

Note that this is just the \( \lambda \)-calculus if we set \( p \) to be a variable.

Rewrite rule:

\[
(\rho) \quad (p \to t) \ u \to [p \ll u].t
\]

Type system: natural extension to simple types for \( \lambda \)-calculus

Representing it all in nets: \( \mathcal{T}([p \ll u].t) \)
Geometry of Interaction

\[ \Pi^\bullet = \begin{bmatrix} \bar{n} & 0 & 0 \\ 0 & 0 & s \\ 0 & s^* & 0 \end{bmatrix} \quad \sigma = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \]

\[ \mathcal{E}\mathcal{X}(\Pi^\bullet, \sigma) = (1 - \sigma^2)\left[\Pi^\bullet \sum_{i=0}^{\infty} (\sigma \Pi^\bullet)^i\right](1 - \sigma^2) \]

Works very well for \( \lambda \)-calculus
Geometry of Interaction for TRS

- TRS: $l \rightarrow r$
  paths in $l$ have nothing to do with paths in $r$ (i.e. the terms are not related in any way.

- $(\lambda x.t)u \rightarrow t[u/x]$
  paths in rhs were already present in the lhs.

How can we combine these ideas together?

$\rho$-calculus: $(l \rightarrow r)t \rightarrow [t \ll l]r$

It’s just $\lambda$-calculus, but add some new operators for patterns:
actually, multiplicatives $+$ constants suffice
Conclusion

- Graphs can represent systems with binders
- Problems of names and freshness have disappeared
- Eliminate a lot of equations and rules that are caused by the textual syntax: properties become identities
- Rewriting calculus extends ideas to a rich setting

Future Work:

- Draw more diagrams
- Geometry of interaction machine for $\rho$-calculus
- Nominal rewriting