Developments in pattern calculus presented at the Third Workshop on the Rho-calculus, King's College, London

Barry Jay

Microsoft Research Ltd

University of Technology, Sydney

23rd October, 2006

Abstract

In lambda-calculus, functions are first-class entities, able to be passed as parameters or returned as results. In rho-calculus, rewrite rules are also first-class. Rules are functions represented by a

pattern (the left-hand side of the rule) and a body (the right-hand side). In pattern calculus, patterns are first-class. That is, dynamic behaviour not only assembles complex procedures from given rewriting rules, but also constructs the rewriting rules themselves dynamically. This is particularly useful for modelling queries, as applied to databases, and indeed for querying XML and other distributed data, where the required patterns must be constructed from distributed information.

This lecture will survey some of the developing theory of pattern calculus. Topics will include: the pure pattern calculus; modelling relational queries; and the relationship to rho-calculus.

Topics

- To outline my research program
- To present the latest account of the pure pattern calculus
- To give examples from query languages.
- To compare this with ρ-calculus, especially as in PPTS.
- To present a polymorphically typed pattern calculus.

Outline my research program

Collaborators: Daniele Gorla, Freeman Huang, Simon Peyton-Jones, Delia Kesner, Helen Lu, Eugenio Moggi, Clara Murdaca, Tony Nguyen, David Skillicorn, ...

I'm drafting a monograph, provisionally entitled *The pattern calculus: computing with functions and structures*. Let's look at the table of contents.

Pure pattern calculus

The history of pattern calculus in syntax

$$t ::= x \mid c \mid t\ t \mid p \to t \mid t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid \mathsf{fix}\ t \text{ (TOPLAS)}$$

$$t ::= x \mid \bullet \mid t\ t \mid t \to_\theta t \text{ (ESOP'06)}$$

$$t ::= x \mid t\ t \mid t \to_\theta t \text{ (my web-site)}$$

$$t ::= x \mid t\ t \mid [\theta]t \to t \text{ (latest).}$$

Pure pattern calculus

The *terms* of the *pure pattern calculus* are given by

$$\begin{array}{lll} t & ::= & \\ & & x \qquad \text{(variable)} \\ & & t\ t \qquad \text{(application)} \\ & & [\theta]\ t \to t \qquad \text{(case).} \end{array}$$

whee θ is a sequence of distinct variables.

Examples

$$\mathsf{elim} = [x]\, x \longrightarrow [y]\, x\ y \longrightarrow y$$

$$\begin{aligned} \mathsf{elim\ leaf\ (leaf\ 4)} &\longrightarrow ([y]\, \mathsf{leaf}\ y \longrightarrow y)\ (\mathsf{leaf}\ 4) \\ &\longrightarrow \{\mathsf{leaf}\ 4/[y]\, \mathsf{leaf}\ y\}\ y \\ &= \{4/y\}y = 4 \end{aligned}$$

Matching and substitution are implicit.

Patterns may have free variables as well as binding variables.

Equality

Let $\lambda x.s$ be $[x]\, x \longrightarrow s$.

$$
\begin{aligned}
&\mathsf{equal}\ x = \\
&\quad [\,]x \longrightarrow \mathsf{true} \\
&\mid [y]\, y \longrightarrow \mathsf{false}.
\end{aligned}
$$

.

Here the free variable x is used as the pattern.

This is *pattern polymorphism.*

Two obvious questions

Q: What is the status of the constructors $\mathsf{leaf}, \mathsf{true}$, etc?

A: They are variables that live in a special context γ introduced shortly.

Q: How are cases combined (using $\mid$)?

A: In the pure calculus, such extensions can be encoded as cases by adopting a special convention for match failure, introduced shortly.

Searching

$$
\begin{aligned}
&\mathsf{select}\ t\ z = \\
&\quad (\mathsf{if}\ t\ z\ \mathsf{then}\ \mathsf{cons}\ z\ \mathsf{else}\ \lambda y.y) \\
&\quad [x, y]\, x\ y \rightarrow (\mathsf{select}\ t\ x)@(\mathsf{select}\ t\ y) \\
&\ \mid [x]\, x \rightarrow \mathsf{nil}
\end{aligned}
$$

where @ is append. For example,

$$\mathsf{select}\ \mathsf{iseven}\ (\mathsf{Node}\ 2\ (\mathsf{Node}\ 3\ (\mathsf{leaf}\ 5)\ (\mathsf{leaf}\ 6)))$$

reduces to $\mathsf{cons}\ 2\ (\mathsf{cons}\ 6\ \mathsf{nil})$.

This is *path polymorphism*.

Constructors

Let γ be a sequence of variables, the *constructors*.

For now, you can think of γ as a fixed class of terms, but soon it will vary.

Atoms and compounds

select is an example of a pattern-matching function with two cases: one for atoms and one for compounds. The latter match the pattern $[x, y]\, x\ y \to \ldots$.

The γ-data structures are given by

$$d ::= x\ (\text{in } \gamma) \mid d\ t$$

For example, if $\mathsf{leaf} \in \gamma$ then $\mathsf{leaf}\ \mathsf{nil}$ and $\mathsf{leaf}\ ([x]x \to x)$ are γ-data structures. Data structures which are applications are *compounds*.

The *γ-matchable forms* are the γ-data structures and the cases. Those which are not compounds are *atoms*.

Matching

$$
\begin{aligned}
\{u \mathbin{/\!\!/} [\theta]\, x\}_\gamma &= \{u/x\} \text{ if } x \in \theta \\
\{c \mathbin{/\!\!/} [\theta]\, c\}_\gamma &= \{\} \text{ if } c \in \gamma \\
\{u\ v \mathbin{/\!\!/} [\theta]\, p\ q\}_\gamma &= \{u \mathbin{/\!\!/} p\}_\gamma \uplus \{v \mathbin{/\!\!/} q\}_\gamma \\
&\quad \text{if } q\ p \text{ is a } \gamma, \theta\text{-matchable form} \\
&\quad \text{and } v\ u \text{ is a } \gamma\text{-matchable form} \\
\{u \mathbin{/\!\!/} [\theta]\, p\}_\gamma &= \mathsf{none} \text{ if } p \text{ is a } \gamma, \theta\text{-matchable form} \\
&\quad \text{and } u \text{ is a } \gamma\text{-matchable form} \\
\{u \mathbin{/\!\!/} [\theta]\, p\}_\gamma &= \text{undefined, otherwise.}
\end{aligned}
$$

Define $\mathsf{none}\ s = \lambda y.y$.

Checking

The *check* of a match μ on a set of variables θ is μ if μ is a substitution whose domain is exactly θ and is none otherwise.

Let p and u be terms and let θ and γ be disjoint sets of variables. Define the *matching* $\{u \,/ [\theta]\, p\}_\gamma$ *of* p *against* u *with respect to binding variables* θ *and constructors* γ to be the check of $\{u \,/\!/ [\theta]\, p\}_\gamma$ on θ.

Each binding variable is bound exactly once.

The match rule

$$([\theta]\, p \to s)\ u >_\gamma \{u\,/[\theta]\, p\}_\gamma\ s \quad \text{(match)}$$

Reduction wrt γ

$$\frac{}{([\theta]\ p \to s)\ u \longrightarrow_\gamma \{u/[\theta]\ p\}_\gamma\ s}$$

$$\ldots$$

$$\frac{p \longrightarrow_{\gamma,\theta} q}{[\theta]\ p \to s \quad \longrightarrow_\gamma \quad [\theta]\ q \to s}$$

$$\ldots$$

When reducing a pattern, its binding variables are treated as if they are constructors.

Motivating example

$$[x]([\,]x \to p)\ x \to x \quad \longrightarrow_{\varepsilon} \quad [x]p \to x$$

since

$$(x \to p)\ x >_x x.$$

Otherwise the pattern $(x \to p)\ x$ would be stuck.

It is safe to match x with itself, to treat it as a constructor, since it is in the pattern used to generate a substitution, not the subject of substitution itself.

A definition of constructors

Constructors are binding variables as they appear in patterns.

This kills two birds with one stone. The pathological example reduces, and the syntactic class of constructors is eliminated.

Names and concurrency

There is a curious parallel with the status of *names* in concurrency. The *concurrent pattern calculus* (with Daniele Gorla) supports symmetric matching

$$([\theta]\, p \to s) \mid ([\varphi]\, q \to t) >_\gamma \{p[\theta|\varphi]q\}(s \mid t)$$

For example,

$$([x]\, c\ 3\ x \to s\ x) \mid ([y]\, c\ y\ \mathsf{true} \to t\ y) \quad >_c \quad s\ \mathsf{true} \mid t\ 3.$$

The constructor c represents a communication channel and x and y are names whose values are exchanged simultaneously (symmetric information exchange).

Parametrising by γ ...

... threatens to destabilise the rewriting theory but we may convert a program p whose constructors are all in γ to the term

$$p \to_\gamma \bullet$$

where $\bullet$ is some closed normal form, e.g. $[x]\, x \to x$. Now reduce using $\longrightarrow_\varepsilon$.

Theorem 0.1 *Reduction with respect to $\gamma = \varepsilon$ is a rewrite relation.*

Proof If $p \longrightarrow_\gamma p'$ then $p \longrightarrow_{\gamma,\theta} p'$ so $[\theta]p \to s \longrightarrow_\gamma [\theta]\, p' \to s$. □

Confluence and progress

Reduction in pure pattern calculus is confluent.

Standard parallel reduction techniques apply.

Note that there is no need to restrict the class of patterns since matching on applications is restricted to compounds, where the applicaiton cannot reduce.

Reduction does not get stuck since normal forms are always matchable forms.

Extensions

The idea: match success or failure provides a branching mechanism.

$$[\theta]p \to s \mid r = \lambda x.([\theta]\, p \to \lambda y.s)\ x\ (r\ x)$$

If p matches against x then $r\ x$ is discarded, else match failure replaces $\lambda y.s$ by the identity, which is then applied to $r\ x$.

Note: this trick will not work in a typed setting.

Updating

There are several forms of updating, just as there are of selecting. Here is one.

$$
\begin{aligned}
&\mathsf{update}\ x\ f = \\
&\quad [y]\, x\ y \to x\ (f\ y) \\
&\mid [y, z]\, y\ z \to \mathsf{update}\ x\ f\ y\ (\mathsf{update}\ x\ f\ z) \\
&\mid [y]\, y \to y.
\end{aligned}
$$

apply2all

Here is another form of updating

$$
\begin{array}{l}
\mathsf{apply2all}\ f = \\
\quad [x, y]\, x\ y \rightarrow f\ (\mathsf{apply2all}\ f\ x)\ (\mathsf{apply2all}\ f\ y) \\
\quad \mid f
\end{array}
$$

Comparison with ρ-calculus

	ρ-calculus	pattern calculus
first-class	rewriting rules	patterns
examples	theorem-proving	queries
eq'ns/constr'nts	yes	no
compounds	no	yes
constructors	primitive	defined
for confluence	rigid patterns	all patterns
matching	explicit	implicit*
...on abstractions	yes	no*

Matching on cases – under development

If desired, add the rule

$$\{\{[\varphi]\, u \to v \mathbin{/\!\!/} [\theta]\, [\varphi]\, p \to q\}_\gamma = \{u \mathbin{/\!\!/} [\theta]\, p\}_{\varphi,\gamma} \uplus \{v \mathbin{/\!\!/} [\theta]\, q\}_{\varphi,\gamma}$$

What are the interesting examples?

Explicit operations – under development

(in discussions with Eugenio Moggi)

Define

$$\{u/x\}s = ([x]\, x \to s)\ u.$$

Then explicit substitution is as usual:

$$\begin{array}{rcl}
\{u/x\}x & >_\gamma & u \\
\{u/x\}y & >_\gamma & y \text{ if } y \neq x \\
\{u/x\}(r\ s) & >_\gamma & \{u/x\}r\ \{u/x\}s \\
\{u/x\}([\theta]\, p \to s) & >_\gamma & [\theta]\, \{u/x\}p \to \{u/x\}s \\
 & & \text{if } x, \mathsf{ftv}(u) \cap \theta = \{\}.
\end{array}$$

Explicit matching

$$
\begin{array}{rcl}
([\,]\,x \to s)\ x & >_\gamma & s \text{ if } x \in \gamma \\
([\theta]\,p\ q \to s)\ (u\ v) & >_\gamma & ([\theta_1]\,p \to ([\theta_2]\,q \to s)\ v)\ u \\
 & & \text{if } p\ q \text{ is a } \theta,\gamma\text{-data structure} \\
 & & \text{and } u\ v \text{ is a } \gamma\text{-data structure} \\
 & & \text{and } \theta_1 = \theta \cap \mathsf{ftv}(p) \text{ and } \theta_2 = \theta \backslash \theta_1 \\
([\theta]\,p \to s)\ u & > & [y]\,y \to y \\
 & & \text{if } p \text{ is a } \theta,\gamma\text{-matchable form} \\
 & & \text{and } u \text{ is a } \gamma\text{-matchable form} \\
([\theta]\,p \to s)\ u & > & \text{undefined, otherwise.}
\end{array}
$$

Some slogans

Interestingly, the types are essentially those of System **F**, the terms are not so different from those of ρ-calculus, and the proof techniques are all standard. What has changed is the viewpoint.

- Patterns are first-class
- Data structures are either atoms or compounds.
- Special cases have special types

Collaborations

1. Freeman Huang, Barry Jay, and David Skillicorn. Programming with heterogeneous structures: Manipulating xml data using **bondi**. (ACSC2006)
2. Freeman Huang, Barry Jay, and David Skillicorn. Adaptiveness in well-typed Java bytecode verification CASCON, 2007.
3. Barry Jay and Delia Kesner. Pure pattern calculus. (ESOP'06) and revised version.
4. C.B. Jay, H.Y. Lu, and Q.T. Nguyen. The polymorphic imperative. (CATS'04)
5. Clara Murdaca and Barry Jay. A relational account of objects (ACSC'06).

Some other papers

1. C.B. Jay. Distinguishing data structures and functions: the constructor calculus and functorial types (TLCA'01).
2. C.B. Jay. Methods as pattern-matching functions. (FOOL'04)
3. C.B. Jay. The pattern calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS,2004)*
4. C.B. Jay. Typing first-class patterns. (HOR'06).
5. C.B. Jay. Type variables simplify sub-typing. (HOR'06.

plus recent submissions on: typed pattern calculus, strong normalisation, and concurrent pattern calculus.

Note: my web-site is not quite up-to-date.

Languages

`www-staff.it.uts.edu.au/~cbj/bondi`, 2005.

and ongoing work.

A patterned worldview