Pure Bondi with datatypes and patterns

Thomas Given-Wilson

June 12, 2007

Background

- Original Bondi code by Jay (2004).
- Pure pattern calculus by Jay and Kesner (2005).
- The role of data structures is not well described in many calculi.
- There are four calculi that explore the role of data structures in the context of pattern matching.

The project goals

- Create modes of operation in Bondi
- Remove the types from some Bondi operational modes
- To implement four calculi as different operational modes
- Use implementaton to check the theory
- Explore the relationship with LISP

Structure of the presentation

- Data structures and the compound calculus - Demo
- Pattern matching and the static pattern calculus - Demo
- Dynamic patterns with the pure pattern calculus - Demo
- Conclusions and Future Work

Traditional Data Structures

There are many forms data structures, consider:

- Pairs: `pair` x y
- Lists: `cons` h t | `nil`
- Trees: `node` x_1 x_2 ... x_n | `leaf` y

However, they are all built from the same structural foundations: atoms and compounds.

Generic Data Sructures

- **Atoms:**
  - x
  - `nil`
- **Compounds:**
  - `pair` x y
  - `cons` h t
  - `node` x_1 x_2 ... x_n
  - `leaf` x

The Compound Calculus

Handle any data structure by focusing on atoms and compounds.

- Is something a compound: `ispair` ("ispair?" in LISP)
- Splitting a compound: `car` `cdr`
- Identifying data structures by constructor: `eqcons` ("eq?" in LISP)

Compound calculus in Bondi

- First and second projections of a compound:
  - `car;;`
  - `cdr;;`
- Head and tail of a list:
  - `let head = fun x -> cdr (car x);;`
  - `let tail = fun x -> cdr x;;`
- Is something a list:
  - `let islist = fun x -> eqcons cons (car (car x));;`

And something a little tricky, the `safehead` function: `let safehead = fun x -> (eqcons nil x) (x) ((ispair x) ((ispair (car x)) ((eqcons cons (car (car x))) (cdr (car x)) ("Not a list.")) ("Not a list.")) ("Not a list."));;`

Pattern Matching theory

Pattern matching in theory:

- $ispair \;=\; u\ v \longrightarrow true \mid u \longrightarrow false$
- $car \;=\; u\ v \longrightarrow u$
- $cdr \;=\; u\ v \longrightarrow v$
- $eqcons\ \hat{x} \;=\; \hat{x} \longrightarrow true \mid \hat{y} \longrightarrow false$

Restriction on patterns: must be data structures.

Static Patterns in Bondi

We can write pattern matching functions:

- ```
  let ispair = | \x \y -> ``True''
               | \z -> ``False'';;
  ```
- ```
  let car = | \x \y -> x;;
  ```
- ```
  let cdr = | \x \y -> y;;
  ```
- ```
  let eqconsnil = | nil -> ``True''
                  | \z -> ``False'';;
  ```

Much cleaner syntax for complex pattern matching:

```
let safehead = | nil -> nil
               | cons \h \t -> h
               | \z -> ``Not a list.'';;
```

Difficulties with variables in the theory

The binding of variables has two complications:

- Binding of variables is immediate
- Scoping is non-trivial

We have the solution by explicitly declaring the binding variables.
So now we can write eqcons:
$eqcons = [x]\hat{x} \rightarrow []x \rightarrow \mathtt{true} \mid [y]\hat{y} \rightarrow \mathtt{false}$

Anything can be a pattern in Bondi

So now anything can be a pattern and we can write our eqcons function:

```
let eq = | \x -> | x -> ‘‘True’’
                 | \y -> ‘‘False’’
```

And some other functions with dynamic patterns:

```
let elim = | \x -> | x \y -> y
                   | \z -> z
letrec update = | \p -> | \f ->
| p \x -> p (f x)
| \u \v -> (update p f u) (update p f v)
| \z -> z;;
```

Conclusions

- No problems in the theory.
- Dynamic mode switching in Bondi.
- Pure (untyped) Bondi.
- Interpreters for four calculi:
  1. Lambda calculus
  2. Compound calculus
  3. Static pattern calculus
  4. Dynamic pattern calculus
- Bondi Zero.

Future Work

- Typed Bondi for each calculus.
- Updating Bondi for the full pattern calculus.
- Type inference.
- Objects and object orientation in Bondi.
- Subtypes and inheritance in Bondi.
- A pattern calculus based programming language.

Contemporary work:

- Barry Jay is refining the pattern calculus theory in many areas mentioned above.
- Matt Roberts is developing a compiler for the pattern calculus.
- Clara Murdaca is working on linking Bondi to databases.

Bibliography

Jay, C. B. 2004, The pattern calculus, *ACM Transactions on Programming Languages and Systems (TOPLAS)* **26**(6), 911–937.

Jay, C. B. 2007, Untyped pattern calculus: Compound calculus. Retrieved 20 April 2007, from http://www-staff.it.uts.edu.au/ cbj/draft-book/pc_sem02.pdf.

Jay, C. B. and Kesner, D. 2005, Pure pattern calculus. Retrieved 10 April 2006, from www.staff.it.uts.edu.au/cbj/Publications/purepatterns.pdf.