Property Based Dispatch in Functional Languages

Christopher Chedeau

LRDE
Laboratoire de Recherche et Développement d'EPITA

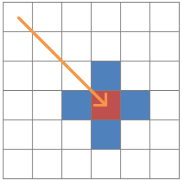
January 18, 2012

http://lrde.epita.fr/

Olena Properties

Type	Name	Values
image	dimension	any, one_d, two_d, three_d

Shift Algorithm



	definition			
	any	unique	multiple	varying
Specialization (1)		✓		
Specialization (2)			✓	

	size	
	any	fixed
Specialization (1)		✓
Specialization (2)	✓	✓

C++ Implementation

```
shift(Window<W>& win, mln_dpsite(W)& dp) {
  // Dispatch on definition property
  shift_(mln_trait_window_definition(W)(), exact(win), dp);
}

shift_(trait::window::definition::unique,
       W& win, mln_dpsite(W)& dp) {
  /* Specialized implementation (1) */
}

shift_(trait::window::definition::multiple,
       W& win, mln_dpsite(W)& dp) {
  /* Specialized implementation (2) */
}
```

Shift Algorithm

	definition			
	any	unique	multiple	varying
Specialization (1)	×	✓	×	×
Specialization (2)	×	×	✓	×

	size	
	any	fixed
Specialization (1)	×	✓
Specialization (2)	✓	✓

C++ Implementation

```
shift(Window<W>& win, mln_dpsite(W)& dp) {
  mlc_is_not(mln_trait_window_definition(W),
            trait::window::definition::any)::check();
  mlc_is_not(mln_trait_window_definition(W),
            trait::window::definition::varying)::check();

  shift_(mln_trait_window_definition(W)(), exact(win), dp);
}

shift_(trait::window::definition::unique,
       W& win, mln_dpsite(W)& dp) {
  mlc_is(mln_trait_window_size(W),
            trait::window::size::fixed)::check();
}
```

Lisp Implementation

```
(defmethod shift (
  (win window)
  (dp dpsite))
  ; Specialization (1)
)
```

```
(defmethod shift (
  (win window)
  (dp dpsite))
  ; Specialization (2)
)
```

Lisp Implementation

```
(defalgo shift (
  (win
    :properties (
      :definition :unique
      :size :fixed)
    window)
  (dp dpsite))
  ; Specialization (1)
)
```

```
(defalgo shift (
  (win
    :properties (
      :definition :multiple)
    window)
  (dp dpsite))
  ; Specialization (2)
)
```

Implementation Overview

```
shift
________________________________________________________________
#1 | implementation: (lambda ...)
   |
   | conditions: | arg1: | (instance-of window)
   |             |       | (has-property :support :regular)
   |             |       | (has-property :definition :unique)
   |             |       | (has-property :size :fixed)
   |             |
   |             | arg2: (instance-of dpsite)

#2 | implementation: (lambda ...)
   |
   | conditions: | arg1: | (instance-of window)
   |             |       | (has-property :support :regular)
   |             |       | (has-property :definition :multiple)
   |             |
   |             | arg2: (instance-of dpsite)
```

Fibonacci

```
(defalgo fibo ((n
    (lambda (n) (< n 2))))
  n)
```

Fibonacci

```
(defalgo fibo ((n
      (lambda (n) (< n 2))))
  n)
```

```
(defun <2 (n) (< n 2))
(defalgo fibo ((n #'<2))
  n)
```

Fibonacci

```
(defalgo fibo ((n
    (lambda (n) (< n 2))))
  n)

(defun <2 (n) (< n 2))
(defalgo fibo ((n #'<2))
  n)
```

```
(defun is (a)
  (lambda (b)
    (eq a b)))

(defalgo fibo ((n (is 0)))
  0)
(defalgo fibo ((n (is 1)))
  1)
```

Fibonacci

```
(defalgo fibo ((n
    (lambda (n) (< n 2))))
  n)
```

```
(defun <2 (n) (< n 2))
(defalgo fibo ((n #'<2))
  n)
```

```
(defun is (a)
  (lambda (b)
    (eq a b)))
```

```
(defalgo fibo ((n (is 0)))
  0)
(defalgo fibo ((n (is 1)))
  1)
```

```
(defalgo fibo (n)
  (+ (fibo (− n 2)) (fibo (− n 1))))
```

Javascript Full Dispatch

```
fibo = FullDispatch()

fibo.add [(n) -> n < 2],
  (n) -> n

fibo.add [null],
  (n) -> fibo(n - 1) + fibo(n - 2)
```

Python Decorators

```
@dispatch(inside(0, 1))
def fibo(n):
    return n

@dispatch(int)
def fibo(n):
    return fibo(n − 1) + fibo(n − 2)
```

Haskell Pattern Matching

```
fibo 0 = 0
fibo 1 = 1
fibo n = fibo (n − 1) + fibo (n − 2)
```

MOP

```
(defmethod fibo (n)
  (+ (fibo (− n 1)) (fibo (− n 2))))

(defmethod fibo ((n (eql 1)))
  n)

(defmethod fibo ((n (eql 0)))
  n)
```

Filtered Dispatch

```
(defun state (n)
  (if (< n 2)
    'terminal
    'general))

(defmethod fibo :filter :state ((n (eql 'terminal)))
  n)

(defmethod factorial :filter :state ((n (eql 'general)))
  (+ (fibo (− n 1)) (fibo (− n 2))))
```

Multimethod.js

```
fibo = multimethod()
  .dispatch((n) -> if n < 2
      'terminal'
    else
      'general')
  .when('terminal',
    (n) -> n)
  .when('general',
    (n) -> fibo(n - 1) + fibo(n - 2))
```

Conclusion

- Conclusion

Questions ?