Adding functional style pattern matching features to object oriented languages

Joseph Thelen

Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

November 12, 2016

Outline

- What Is Pattern Matching?
- Examples in Haskell
- Adding Pattern Matching to Java
- Adding Pattern Matching to C++
- Recap

What is pattern matching?

"Pattern matching is the act of checking a given sequence of tokens for the presence of the constituents of some pattern."

— Wikipedia

What is pattern matching?

"Pattern matching is the act of checking a given sequence of tokens for the presence of the constituents of some pattern."

Awesome Haskell Example - Simple Function

```
ourFunction x = x + 1
```

Awesome Haskell Example - Simple Function

```
ourFunction x = x + 1
```

Awesome Haskell Example - Simple Function

```
ourFunction x = x + 1
```

Awesome Haskell Example - Simple Function

```
ourFunction x = x + 1
```

Awesome Haskell Example - Simple Function

```
ourFunction x = x + 1
```

Awesome Haskell Example - Simple Function

```
ourFunction 0 = -1
ourFunction x = x + 1
```

Awesome Haskell Example - Simple Function

```haskell
ourFunction 0 = -1
ourFunction x = x + 1
```

```
> ourFunction 12
13
```

Awesome Haskell Example - Simple Function

```haskell
ourFunction 0 = -1
ourFunction x = x + 1
```

```
> ourFunction 12
13

> ourFunction 0
-1
```

Awesome Haskell Example - Simple Function

```
ourFunction 0 = -1
ourFunction x = x + 1
```

```
> ourFunction 12
13

> ourFunction 0
-1

> ourFunction -10
-9
```

Awesome Haskell Example - Now with lists!

```
ourFunction [] = -1
ourFunction (0:xs) = sum xs
ourFunction (x:xs) = x * sum xs
```

Awesome Haskell Example - Now with lists!

```
ourFunction [] = -1
ourFunction (0:xs) = sum xs
ourFunction (x:xs) = x * sum xs
```

"cons" - Constructing a list
e.g. `(5:[10, 15])` becomes `[5, 10, 15]`

Decomposition - breaking up data structures based on a pattern

Awesome Haskell Example - Now with lists!

```
ourFunction [] = -1
ourFunction (0:xs) = sum xs
ourFunction (x:xs) = x * sum xs
```

```
> ourFunction [2, 2, 3, 4, 5]
28
```

Awesome Haskell Example - Now with lists!

```
ourFunction [] = -1
ourFunction (0:xs) = sum xs
ourFunction (x:xs) = x * sum xs
```

```
> ourFunction [2, 2, 3, 4, 5]
28

> ourFunction []
-1
```

Awesome Haskell Example - Now with lists!

```
ourFunction [] = -1
ourFunction (0:xs) = sum xs
ourFunction (x:xs) = x * sum xs
```

```
> ourFunction [2, 2, 3, 4, 5]
28

> ourFunction []
-1

> ourFunction [0, 1, 2, 3]
6
```

Awesome Haskell Example - Adding some recursion

```
ourFunction [] = -1
ourFunction (0:xs) = ourSum xs
ourFunction (x:xs) = x * ourSum xs

ourSum [] = 0
ourSum (x:xs) = x + ourSum xs
```

Awesome Haskell Example - Adding some recursion

```
ourFunction [] = -1
ourFunction (0:xs) = ourSum xs
ourFunction (x:xs) = x * ourSum xs

ourSum [] = 0
ourSum (x:xs) = x + ourSum xs
```

Awesome Haskell Example - Adding some recursion

```
ourSum [] = 0
ourSum (x:xs) = x + ourSum xs
```

```
ourSum [1, 2, 3]
```

Becomes

```
1 + ourSum [2, 3]
```

Awesome Haskell Example - Adding some recursion

```haskell
ourSum [] = 0
ourSum (x:xs) = x + ourSum xs
```

---

ourSum [1, 2, 3]

Becomes

```
1 + ourSum [2, 3]
        2 + ourSum [3]
```

Awesome Haskell Example - Adding some recursion

```
ourSum [] = 0
ourSum (x:xs) = x + ourSum xs
```

```
ourSum [1, 2, 3]
```

Becomes

```
1 + ourSum [2, 3]
        2 + ourSum [3]
                3 + ourSum []
```

Awesome Haskell Example - Adding some recursion

```haskell
ourSum [] = 0
ourSum (x:xs) = x + ourSum xs
```

```
ourSum [1, 2, 3]
```

Becomes

```
1 + ourSum [2, 3]
        2 + ourSum [3]
                3 + ourSum []
                        0
```

Adding pattern matching to Java

The *OOMatch* Project

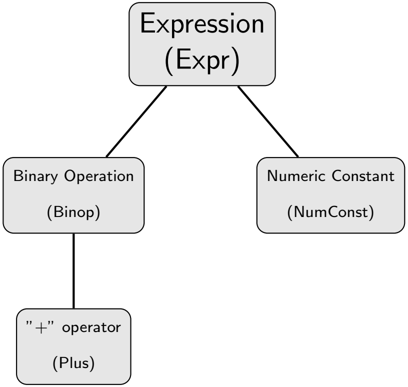
“Pattern Matching as Dispatch in Java”

Adam Richard, Ondȓej Lhoták
University of Waterloo

Pattern matching in *OOMatch* - Implementation

- Modification to the Java compiler (An *extension*)
- Patterns are represented as objects

OOMatch example - Class Hierarchy



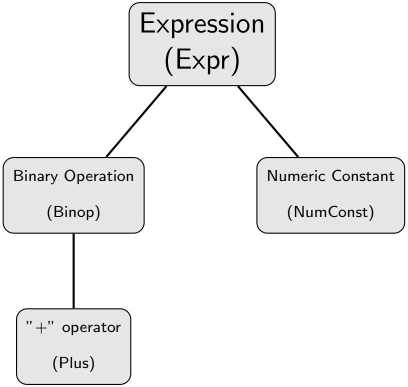
```
//Arithmetic expressions
abstract class Expr { ... }

//Binary operators
class Binop extends Expr { ... }

//'+' operator
class Plus extends Binop { ... }

//Numeric constants
class NumConst extends Expr { ... }
```

OOMatch example - Class Hierarchy



```
(1 + (4 + 3))

Breaks down into:


Plus(
  NumConst,
  Plus(NumConst, NumConst)
)
```

OOMatch example - "Optimizer"

```
//do nothing by default
Expr optimize(Expr e) { return e; }

//Anything + 0 is itself
Expr optimize(Plus(Expr e, NumConst(0)))
{ return e; }

//Const folding
Expr optimize(Binop(NumConst c1,
                    NumConst c2) op)
{ return op.eval(c1, c2); }
```

OOMatch example - "Optimizer"

```
//do nothing by default
Expr optimize(Expr e) { return e; }

//Anything + 0 is itself
Expr optimize(Plus(Expr e, NumConst(0)))
{ return e; }

//Const folding
Expr optimize(Binop(NumConst c1,
                    NumConst c2) op)
{ return op.eval(c1, c2); }
```

OOMatch example - "Optimizer"

```
//do nothing by default
Expr optimize(Expr e) { return e; }

//Anything + 0 is itself
Expr optimize(Plus(Expr e, NumConst(0)))
{ return e; }

//Const folding
Expr optimize(Binop(NumConst c1,
                    NumConst c2) op)
{ return op.eval(c1, c2); }
```

OOMatch example - "Optimizer"

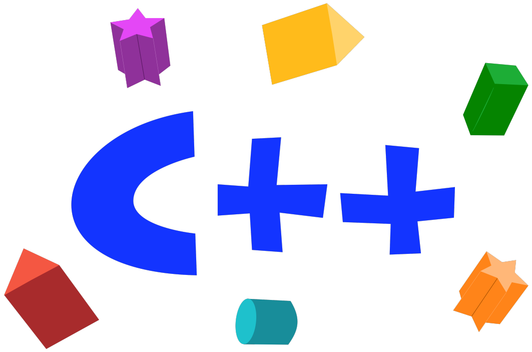
```
//do nothing by default
Expr optimize(Expr e) { return e; }

//Anything + 0 is itself
Expr optimize(Plus(Expr e, NumConst(0)))
{ return e; }

//Const folding
Expr optimize(Binop(NumConst c1,
                    NumConst c2) op)
{ return op.eval(c1, c2); }
```

```
(1 + (4 + 3))       --> Do nothing
(1 + 2)             --> Constant folding
(1 + (2 + 5)) + 0   --> Anything + zero
```

Adding pattern matching to C++



The *Mach7* project

"Open Pattern Matching for C++"

Yuri Solodkyy, Gabriel Dos Reis, Bjarne Stroustrup
Texas A&M University

Pattern matching in *Mach7* - Implementation

- Additions are made as *libraries*
- Patterns represented as *expression templates*

Simple *Mach7* example

```
int factorial(int n) {
 unsigned short m;
 Match(n) {
    Case(0) return 1;
    Case(m) return m * factorial(m-1);
    Case(_) throw std::invalid_argument("factorial");
  }  EndMatch
}
```

Simple *Mach7* example

```
int factorial(int n) {
 unsigned short m;
 Match(n) {
    Case(0) return 1;
    Case(m) return m * factorial(m-1);
    Case(_) throw std::invalid_argument("factorial");
  }  EndMatch
}
```

Recap

OOMatch (Java)

- "Patterns as objects"
- Work done at runtime
- Pattern matching as dispatch

Mach7 (C++)

- Patterns as *expression templates*
- Work done at compile time
- No pattern matching as dispatch

The End

Thanks for attending today!

Thanks to the faculty and reviewer(s) for all the constructive feedback, and to my roommates for putting up with me this semester.

Questions?

Contact:
thele116@morris.umn.edu

References

Solodkyy, Yuriy and Dos Reis, Gabriel and Stroustrup, Bjarne.
Open Pattern Matching for C++.
In proceedings GPCE 2013, pages 33-42, Indianapolis, Indiana, USA.

Richard, Adam and Lhotak, Ondrej.
OOMatch: Pattern Matching As Dispatch in Java.
In proceedings OOPSLA 2007, pages 771-772, Montreal, Quebec, Canada.

See paper for additional references.