

## Predicate Dispatching: A Unified Theory of Dispatch

Michael Ernst, Craig Kaplan,  
and Craig Chambers

University of Washington  
Seattle, Washington, USA  
<http://www.cs.washington.edu/research/projects/cecil>

Ernst, ECOOP '98, page 1

## Dispatching

Select one *case* from a *generic function*

- object-oriented dispatch, including multi-methods
- classifiers, predicate classes
- pattern matching

Case selection

- applicability
- overriding

Static checking

- completeness
- uniqueness

Ernst, ECOOP '98, page 2

## Goals

Unify and generalize dispatching mechanisms  
Small, orthogonal basic model  
Functions are extensible  
Static checking

Solution: predicate dispatching

Ernst, ECOOP '98, page 3

## Outline

Predicate dispatching  
Examples  
Semantics

Ernst, ECOOP '98, page 4

## Applicability

Predicates are boolean formulas over class tests  
and program expressions

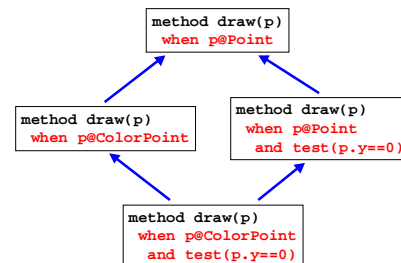
```
pred ::= expr @ class
      | test(expr)
      | let var := expr
      | not pred
      | pred or pred
      | pred and pred
```

```
method draw(p)
  when p@Point and test(p.y == 0)
  { draw point on x axis }
```

Ernst, ECOOP '98, page 5

## Overriding

Overriding is determined by logical implication



Ernst, ECOOP '98, page 6

## Object-oriented dispatch (SmallTalk, C++, Java)

Applicability: run-time class is subclass of  
specializer

Overriding: subclassing (most specific  
specializer)

```
class Point;
method draw(p) when p@Point { ... };

class ColorPoint extends Point;
method draw(p) when p@ColorPoint { ... };

Equivalently: method draw(p@Point) { ... };
               method draw(p@ColorPoint) { ... };
```

Ernst, ECOOP '98, page 7

## Multi-method dispatch (CLOS, Cecil, Dylan)

Applicability: run-time classes are subclasses  
of specializers

Overriding: subclassing over all specializers

```
method equal(p1, p2)
  when p1@Point and p2@Point
  { return p1.x==p2.x and p1.y==p2.y; }

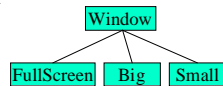
method equal(p1, p2)
  when p1@ColorPoint and p2@ColorPoint
  { return p1.x==p2.x and p1.y==p2.y
    and p1.color==p2.color; }
```

Ernst, ECOOP '98, page 8

## Classifiers (Kea, Cecil)

Applicability: boolean conditions of runtime state

Overriding: lexical order



```
classify(w@Window)
  as FullScreen when test(w.area() == RootWindow.area())
  as Big when test(w.area() > RootWindow.area()/2)
  as Small otherwise;

method move(w) when w@FullScreen { do nothing }
method move(w) when w@Big { move wireframe outline }
method move(w) when w@Small { move opaque window }
```

Ernst, ECOOP '98, page 9

## Pattern matching (ML, Haskell)

Applicability: structural equivalence

Overriding: lexical order

```
fun sumList (nil) = 0
  | sumList (h::t) = h + sumList(t);

method sumList(l) when l@Nil
  { return 0; }
method sumList(l)
  when l@Cons and let h:=l.head and let t:=l.tail
  { return h + sumList(t); }
```

Ernst, ECOOP '98, page 10

## New capability: disjunction

**zip:** given a pair of lists, return a list of pairs

```
method zip(l1, l2)
  when l1@Nil or l2@Nil
  { return Nil; }

method zip(l1, l2)
  when l1@Cons and l2@Cons
  { return Cons(Pair(l1.head, l2.head),
    zip(l1.tail, l2.tail)); }
```

Ernst, ECOOP '98, page 11

## Predicate abstractions

Capabilities of in-line predicates:

- return boolean value
- bind variables

```
predicate On_x_axis(p)
  when (p@CartesianPoint and test(p.y == 0))
  or (p@PolarPoint
    and (test(p.theta == 0) or test(p.theta == pi)))

method draw(p) when p@Point { ... }
method draw(p) when On_x_axis(p) { ... }
```

Ernst, ECOOP '98, page 12

## Run-time behavior

Applicability: evaluate predicates (at run time)  
Optimization: common subexpression elimination  
Overriding: logical implication  
Computed at compile time, used at run time

Ernst, ECOOP '98, page 13

## Typechecking

Completeness:  
*Informally*, disjunction of predicates = true  
Uniqueness: for each pair of predicates,  
– disjoint,  
– one subsumes the other, or  
– their intersection is overridden

Ernst, ECOOP '98, page 14

## Comparing predicates

Compile-time tautology tests  
In general: **undecidable**  
Black-box program expressions: **NP-hard**  
Small predicate expressions: **fast**  
Consistent classes: **complete**

Ernst, ECOOP '98, page 15

## Future work

More efficient implementation strategies  
Separate typechecking

Ernst, ECOOP '98, page 16

## Contributions of predicate dispatching

Generalizes and subsumes previous techniques in a  
common framework  
Enables new varieties of tests  
Implementation:  
<http://www.cs.washington.edu/research/projects/cecil/www/Gud>

Ernst, ECOOP '98, page 17