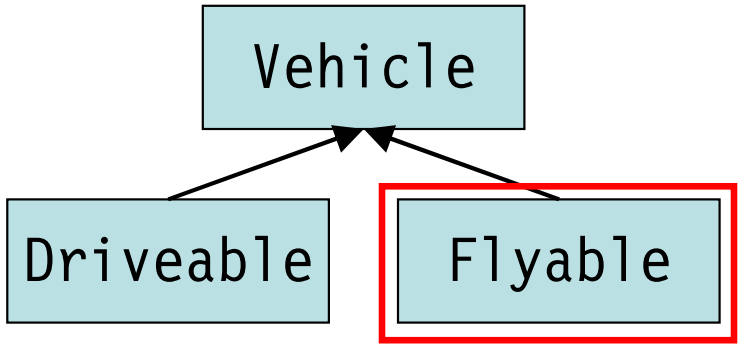
Modularly Typesafe Interface Dispatch in JPred

Christopher Frost and Todd Millstein
University of California, Los Angeles
{frost,todd}@cs.ucla.edu

Background: Multimethod Dispatch

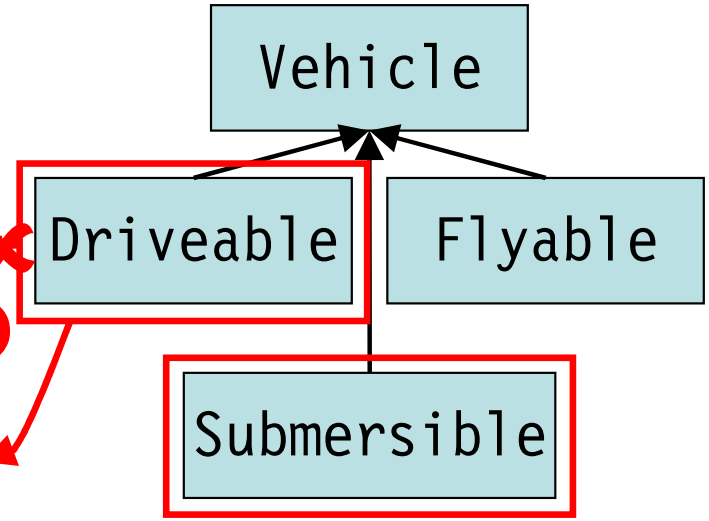
```
class Spy {
   void escape(Vehicle v)                    { … }
   void escape(Vehicle@Driveable v)          { … }
   void escape(Vehicle@Flyable v)            { … }
}
```

```
Vehicle v = ...;
spy.escape(v);
```



Background: Multimethod Dispatch

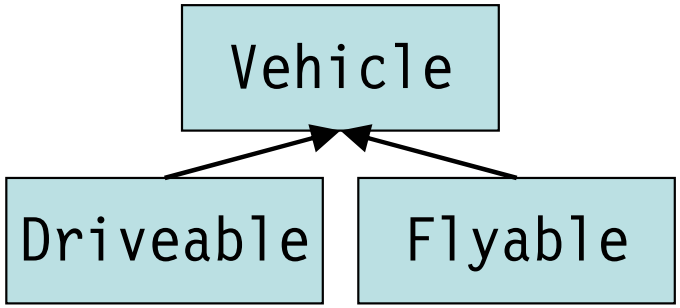
```
class Spy {
  void escape(Vehicle v)
  void escape(Vehicle@Driveable v)
  void escape(Vehicle@Flyable v)
  void escape(Vehicle@Driveable v)
}
```



- Multimethod lookup: most specific method for runtime types of arguments
- Possible message dispatch errors:
  - Message-not-understood
  - Message-ambiguous

Background: Multimethod Dispatch

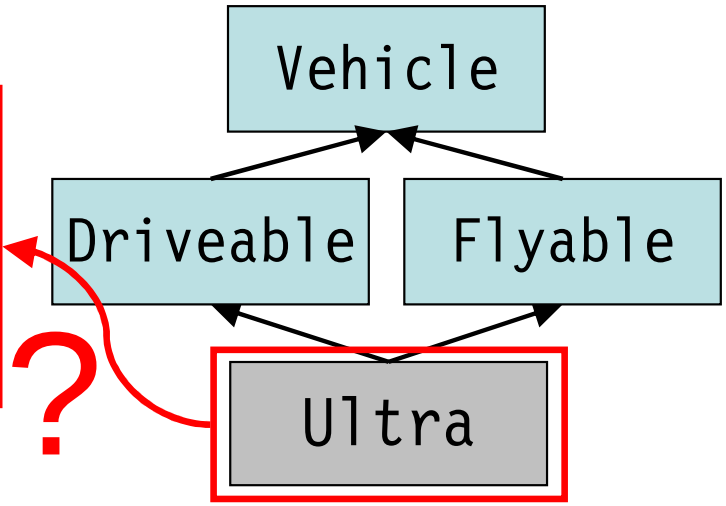
```
class Spy {
  void escape(Vehicle v)
  void escape(Vehicle@Driveable v)
  void escape(Vehicle@Flyable v)
}
```



- Typechecking: ensure methods are exhaustive and unambiguous
- Typechecking can be done modularly
  - Only use knowledge of modularly available types

Dispatch on Java Interfaces

```
class Spy {
  void escape(Vehicle v)
  void escape(Vehicle@Driveable v)
  void escape(Vehicle@Flyable v)
}
```



- Let `Vehicle`, `Driveable`, and `Flyable` be interfaces
- Method lookup can become ambiguous!
- Modular typechecking must conservatively rule out
  - Must rule out multimethods with 2+ interface dispatches
  - Underlying problem: multiple inheritance

Prior Approaches

- No typechecking
  [commonloops, clos, dylan, cmm]
- Require whole program analysis
  [cecil, tuple, doublecpp, nice]
- Forbid interfaces from being dispatched upon
  [multijava, jpred (originally)]
- Restrict multiple inheritance to only within a module
  [dubious, half & half]
- Linearize multiple dispatch or inheritance semantics
  [polyglot, castagna]

Our Contributions

- Practical interface dispatch
  - Preserves modular typechecking
  - Key: predicate dispatch's expressiveness [Ernst et al 98]
- Instantiate in the context of JPred [Millstein 04]
- Formalize and prove type soundness
- Demonstrate utility in practice through case studies
  - JPred compiler
  - Eclipse

Background: JPred

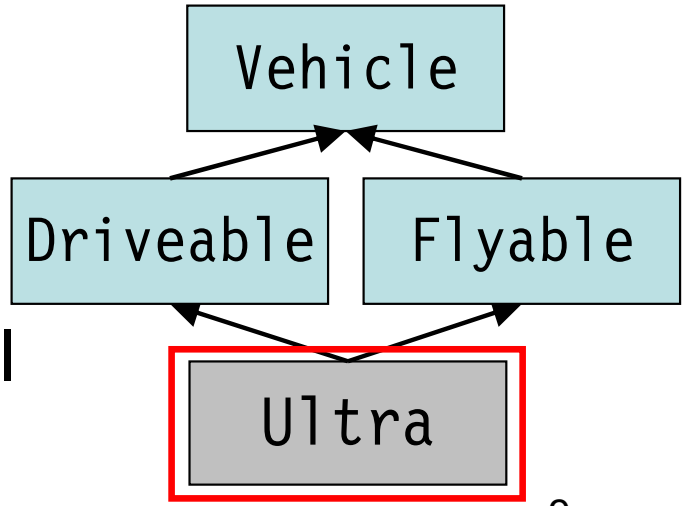
- Method predicate guards subsume multimethod's expressiveness
- Dispatch on class types, fields, linear arithmetic, binding, and &&, ||, and ! operations
- $m_1()$ overrides $m_2()$ if $m_1()$'s predicate implies $m_2()$'s

```
class Spy {
   void escape(Vehicle v)                          { … }
   void escape(Vehicle v) when v@Driveable         { … }
   void escape(Vehicle v) when v@Flyable           { … }
   void escape(Vehicle v)
      when v@Driveable && inSwamp                  { … }
   boolean inSwamp;
}
```

Fixing Interface Dispatch Modularly

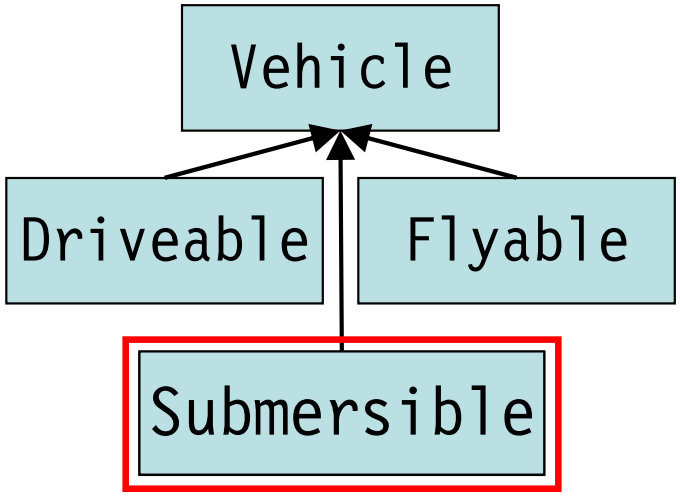
```
class Spy {
  void escape(Vehicle v)
  void escape(Vehicle v) when v@Driveable && !v@Flyable
  void escape(Vehicle v) when v@Flyable
  void escape(Vehicle v) when v@Driveable && v@Flyable
}
```

- Insight: Predicate dispatch allows modular resolution of all possible multiple inheritance ambiguities
- Example resolution approaches:
  - Add additional method(s) to cover all possible ambiguities
  - Change existing method(s) to cover all possible ambiguities



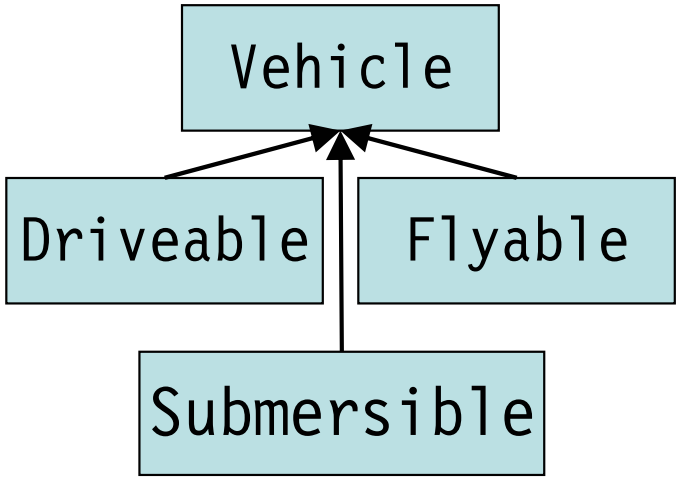
Ordered Dispatch

```
class Spy {
  void escape(Vehicle v)
  void escape(Vehicle v)
    when P1
  void escape(Vehicle v)
    when P2                  && !P1
  void escape(Vehicle v)
    when P3                  && !P1          && !P2
}
```



Ordered Dispatch

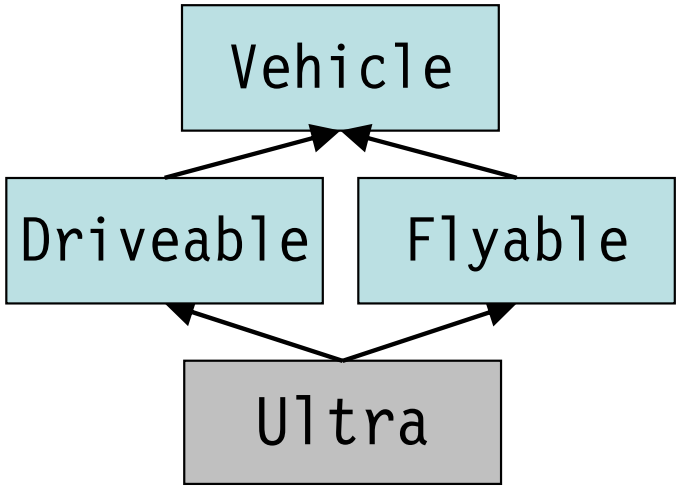
```
class Spy {
   void escape(Vehicle v)
       when v@Flyable { … }
   |   when v@Driveable { … }
   |   when v@Submersible { … }
   |   { … }
}
```



- “First-match” lookup
  - Mirrors common `instanceof` coding styles
- Purely syntactic sugar
- Methods can use combinations of ordered and non-ordered dispatch

Predicate Satisfiability

```
class Spy {
   void escape(Vehicle v)
       when v@Flyable { … }
   |   when v@Driveable { … }
   |   when v@Ultra { … }
   |   { … }
}
```



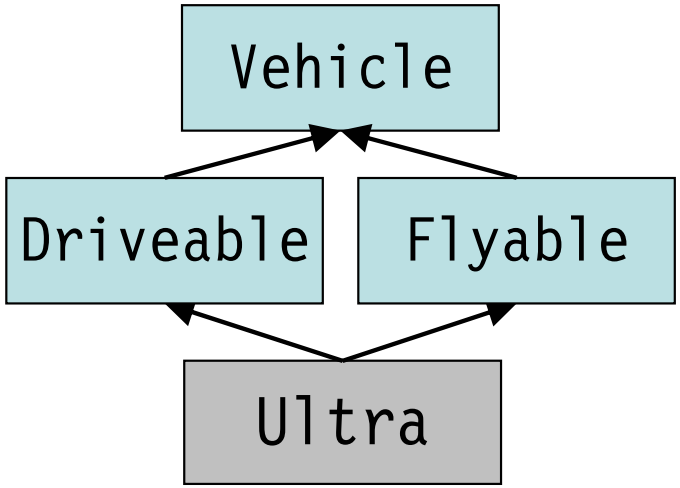
Desugars to

```
void escape(Vehicle v)
   when v@Ultra && !v@Flyable && !v@Driveable { … }
```

- `Ultra`'s predicate is not satisfiable
- Easy mistake with large interface hierarchies

Predicate Satisfiability

```
class Spy {
   void escape(Vehicle v)
        when v@Flyable { … }
   |    when v@Driveable { … }
   |    when v@Ultra { … }
   |    { … }
}
```



Typechecker: unreachable!

- Add static typecheck:
  Unsatisfiable predicates
- Also useful for non-ordered dispatch

Implementation

- JPred: extension to the Polyglot Java compiler
  [Nystrom et al 03]

- Original JPred disallowed interface dispatch

- JPred compiler changes
  - Allowed interface dispatch
  - Added ordered dispatch to the parser
  - Added predicate satisfiability check
  - Code generation unchanged

Featherweight JPred

- Extension of Featherweight Java [Igarashi et al 01]
  - Added interfaces and method predicates
- Formalized syntax and dynamic and static semantics
- Proved a type soundness theorem through progress and preservation
- Validates sufficiency of modular typechecking
- First provably sound formalization of predicate dispatch

Case Studies

- JPred Compiler
  - Originally written in JPred using class dispatch
- Eclipse's Java Development Tooling UI Plugin
  - Originally written in Java

Case Studies: JPred Compiler

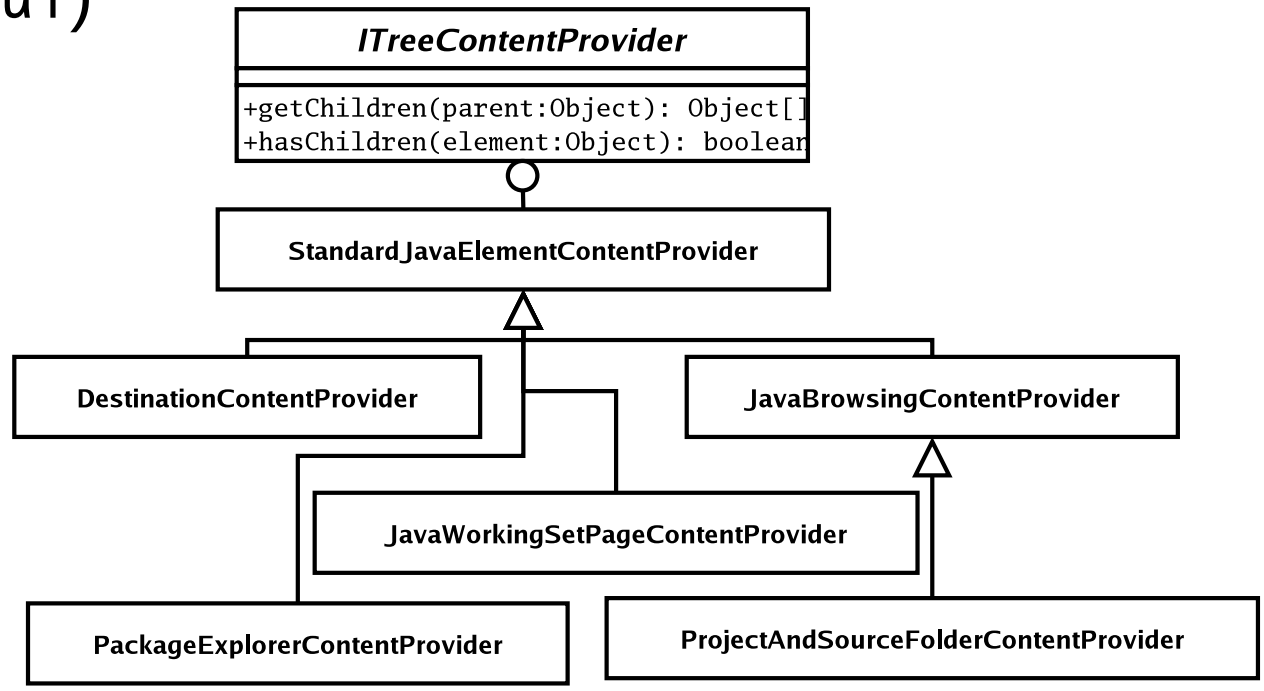
- Originally written as a Polyglot extension in JPred using class dispatch
- Polyglot structure
  - Interface hierarchy representing AST nodes
  - Class hierarchy implementing the interfaces
- Compiler passes using visitor design pattern
  - `instanceof` tests to provide specialized AST behavior
- Polyglot intends for clients to interact only with interfaces

---

- All AST class dispatches now interface dispatches
- 28 updated messages: 14 single, 14 two+ methods
- Unsatisfiable predicate check caught bug in typechecker
- 3% compile time increase

Case Studies: Eclipse JDT UI

- Goal: Evaluate JPred's utility for complex programs not designed for predicate dispatch
- Updated:
  - `ITreeContentProvider` implementors' `getChildren()` `hasChildren()`
  - Eclipse's Java Development Tooling UI Plugin classes (`org.eclipse.jdt.ui`)



Case Studies: Eclipse JDT UI

Eclipse method snippet:

```
public Object[] getChildren(Object parentElement) {
   ...
   if (parentElement instanceof IJavaModel)
      return concatenate(
            super.getChildren(parentElement),
            getNonJavaProjects((IJavaModel)parentElement));

   if (parentElement instanceof IProject)
      return ((IProject)parentElement).members();

   return super.getChildren(parentElement);
   ...
}
```

Case Studies: Eclipse JDT UI

- Reduced type casts (all on interfaces) from 30 to 3
- Errors found by the compiler:
  - 1 error: repeated `instanceof` tests, second test unreachable
  - 1 possible error: series of `instanceof` tests missing `else` clause, not exhaustive
- JPred limitations:
  - Dispatches occur only at “top level”
    - But Eclipse often does applicability tests within `try`/`catch`
  - Method calls are not allowed within predicates
  - JPred style targeted at logically independent cases
    - But Eclipse has instances of fall-through logic

Conclusions

- Demonstrated a practical resolution to the tension between multiple dispatch and multiple inheritance
  - While retaining fully modular static typechecking
- Key idea: predicate dispatch's expressiveness allows programmers to modularly resolve multiple inheritance ambiguities
- Validated our approach:
  - Formalized and proved type soundness
  - Demonstrated utility through two case studies