MASKED TYPES
for Sound Object Initialization

Xin Qi and Andrew C. Myers
Cornell University
Fix the initialization problem

- Current mechanisms for object initialization are unsound
- This talk: a lightweight type system for sound initialization
  - Gets rid of null-pointer exceptions
  - Handles inheritance and cycles
- Implementation – J\mask

Xin Qi
Alice wants a data structure…

A data structure…

...
Alice wants a data structure…

Remember: initialize before use!

Initialization

Invariants established

Normal use

This methodology does not work!

Xin Qi
An example with inheritance

class Point {
  int x, y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
    display();
  }
  void display() {
    System.out.println(x + " " + y);
  }
}

class CPoint extends Point {
  Color c;
  CPoint(int x, int y, Color c) {
    super(x, y);
    this.c = c;
  }
  void display() {
    System.out.println(x + " " + y + " " + c.name());
  }
}
A bug with no one to blame

- Each individual class looks OK
- Classes don’t agree on the initialization contract

```java
class Point {
    int x, y;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
        display();
    }
    void display() {
        System.out.println(x + " " + y);
    }
}

class CPoint extends Point {
    Color c;
    CPoint(int x, int y, Color c) {
        super(x, y);
        this.c = c;
    }
    void display() {
        System.out.println(x + " " + y + " " + c.name());
    }
}
```

Xin Qi

Masked Types
Unsound initialization

- Problem: initialization is *unsound*:
  - Can read uninitialized object fields
- “Solution” (Java/C#): fields pre-initialized with default “null” values
  - Null is a value of all object types
  - Ubiquitous null checks and possible null-pointer exceptions
- Result: unreliable software
Current language support

- Object-oriented initialization is unsound
  - Inheritance
  - Cyclic data structures

- Functional languages trade expressiveness for soundness
  - Cyclic data structures need encoding/refs
- **T \ f**
  - Base type T
  - *Field mask* on f
    - Possibly uninitialized
    - Not readable

- **Assignments remove masks**
  ```
  // x : CPoint \ c
  x.c = new Color("Blue");
  // x : CPoint
  ```

- **Typestates**
More masks

- T \ *
  - Disallows reading any field

- Point \ Point.sub
  - Disallows reading fields declared in subclasses
  - Point \ * = Point \ x \ y \ Point.sub

- Abstract masks for data abstraction
Inheritance

- Make initialization contracts explicit
- Methods and constructors have *mask effects*
  - Capture initialization contracts
  - Support modular type-checking
class Point {
  int x, y;
  Point (int x, int y) {
    this.x = x;
    this.y = y;
    display();
  }
  void display () {
    System.out.println (x + " " + y);
  }
}

class CPoint extends Point {
  Color c;
  CPoint (int x, int y, Color c) {
    super (x, y);
    this.c = c;
  }
  void display () {
    System.out.println (x + " " + y + " " + c.name ());
  }
}

If we blame the Point class, ...
Xin Qi

Masked Types

class Point {
  int x, y;
  Point(int x, int y)
  { 
    this.x = x;
    this.y = y;
    display();
  }
  void display()
  {
    System.out.println(x + " " + y);
  }
}

class CPoint extends Point {
  Color c;
  CPoint(int x, int y, Color c)
  {
    super(x, y);
    this.c = c;
  }
  void display()
  {
    System.out.println(x + " " + y + " " + c.name());
  }
}
Cyclic data structures

- Cyclic data structures are common
  - Doubly-linked lists
  - Circular lists
  - Binary trees with parent pointers

- Sound initialization is challenging
  - Disallow reading fields pointing to "incomplete" objects
  - Know when initialization completes
An example

class Node {
    Node next;
}
Node x = new Node();
Node y = new Node();
x.next = y;
...
y.next = x;

next
next
x ← y

y.next uninitialized
⇒ not safe to read x.next

“ties the knot”
⇒ both objects are safe to use

Conditional masks
    ○ Dependencies between masks
    ○ Graph theory-based type checking
An example

class Node {
    Node next;
}
Node x = new Node();
Node y = new Node();
x.next = y;  
...
y.next = x;

Depends on
x.next ← y.next
Depends on

x : Node \ next[y.next]
y : Node \ next[x.next]

Conditionally masked type

Removal of circular dependencies

x : Node
y : Node
Object calculus with heap
- No special value “null”
- Uninitialized fields cannot be read

Object initialization is sound
- Evaluation never gets stuck
- Proof:
  - Encoding of graph theoretical problems
  - progress + preservation
J\mask language

- Constructors not special
- Default effects reduce annotation burden

Implementation
  - Polyglot compiler framework (Nystrom, Clarkson & Myers 03)
  - Flow-sensitive type system
  - Translation to Java by type erasure
Experience

- Java Collections Framework (1.4.2)
  - LinkedList, ArrayList, HashMap, TreeMap, Stack, …
  - 29 source files, 18,000 LOC

- Results
  - Handled JCF initialization patterns
  - Removed nulls for initialization
  - Low annotation burden
    - 11 explicit effects
    - 11 explicit masked types
Related work

- Non-null types
  - @NonNull annotations (Java 6/7)
  - Delayed types (Fähndrich & Xia 07)

- Typestates
  - Typestates for objects (DeLine & Fähndrich 04)
  - Heap monotonic typestates (Fähndrich & Leino 03)

- Static analysis
  - Detecting null-pointer exceptions (FindBugs)
  - Shape analysis
Summary

- Sound and expressive initialization
  - Handles inheritance and cycles
- Local, modular reasoning
  - Mask effects
  - Abstract masks
- Lightweight
  - Low annotation burden
    - No aliasing information
    - Default annotation
  - No run-time overhead
- Maybe the end of null-pointer exceptions!
J\mask source code available at http://www.cs.cornell.edu/Projects/jmask/