Subtyping, Subclassing, and Trouble with OOP

The Paper

http://okmij.org/ftp/Computation/Subtyping/Trouble.html

- A specific problem with OOP inheritance
- And a specific solution; functional programming
- Wikipedia with other resources Circle Ellipse Problem

The Claim

When code and data are bundled together, inheritance (including implementation of interfaces) produces subclasses that may not be subtypes, and are not necessarily be safe to substitute for one another.

The Setup

- CBag, an interface for a collection of numbers
- Operators depending only on the public interface for Bag
- CSet, which implements `put` to avoid duplicates

Foo

A function using only the public interface which decides if `a + b` is contained in `c`.

```
bool foo(const CBag& a, const CBag& b, const CBag& c)
{
    CBag & ab = *(a.clone());      // Clone a to avoid clobbering it
    ab += b;                       // ab is now the union of a and b
    bool result = ab <= c;
    delete &ab;
    return result;
}
```

Foobar

A change to move ab from the heap to the stack, for efficiency…

```
bool foo(const CBag& a, const CBag& b, const CBag& c)
{
    CBag ab;
    ab += a;                            // Clone a to avoid clobbering it
    ab += b;                            // ab is now the union of a and b
    bool result = ab <= c;
    return result;
}
```

It Gets Worse

- Limited to inheritance, but implementation of an interface is sufficient
- Immutability by itself doesn't help

```
(define (fnb bag)
  (send bag 'put 5)
  (send bag 'put 5)
  (send bag 'count 5))
```

BRules

- No virtual methods or virtual inheritance
- No visible members or methods in any public data structure
- No mutations to public data structures

  The rules break the major tenets of OOP: for example, values no longer have a state that is separate from their identity.

FBag Separates Data and Methods

- Immutable data
- Functions polymorphic in their inputs, always produce an FBag

  It must be stressed that the functions that operate on a FBag are not FBag's methods; in particular, they are not a part of the class FBag, they are not inherited and they cannot be overridden.

FSet Constrains and Extends FBag

```
class FSet : public FBag {
public:
    FSet(void) {}
    FSet(const FBag& bag) : FBag(remove_duplicates(bag)) {}
};

bool memberof(const FSet& set, const int elem)
{ return count(set,elem) > 0; }
```

Inheritance

- An FSet is an FBag, reconsider `foo`
- The caller's FSet declaration will create a properly constrained FBag

  Because all FSet values are created by an FBag constructor, all FBag operations automatically apply to an FSet value.

Comments

- Discussion: uf-integers vs. integers
- Code: circles and squares vs ellipses and rectangles
- BRules allow data types to change class when necessary
- Preserve code sharing between similar classes
- Similarity to typed pure languages