Lecture 14: Nominal and Structural Types

David Bremner

February 28, 2024

Binary Tree Example

p. 165

Recall our algebraic data type from the previous lecture.

```
(define-type BT
  [mt]
  [node (v : Number) (l : BT) (r : BT)])
```

BT in Java 1/2

bt1

```
abstract class BT {
    abstract public int size();
}
class mt extends BT {
    public int size() {
        return 0;
    }
}
```

BT in Java 2/2

bt1

```java
class node extends BT {
    int v;
    BT l, r;
    node(int v, BT l, BT r) {
        this.v = v;
        this.l = l;
        this.r = r;
    }
    public int size() {
        return 1 + this.l.size() + this.r.size();
    }
}
```

BT in Java Discussion

p. 166

Where are the tags/predicates

- People call Java “statically typed”, but (like most OO) it relies on dynamic dispatch.

Subclassing vs. Algebraic data types

- In Java, we can add new variants without editing others
- With ADT, we can add new functions for fixed variants.

Nominal types

Suppose we have a duplicate subclass for `mt`?

```
class empty extends BT {
    public int size() {
        return 0;
    }
}
```

Are they interchangeable?

bt2

```
static int m(mt o) {
    return o.size();
}
```

Structural typing

p. 167

- In structural typing, the type of a class is not its name but rather a description of its fields and methods.
- In `typed/racket`, classes are first class values, and have types.

Structural typing 1/3

DETOUR

- typed/racket infers the type `(Class (init (with-size Real)) (size (-> Real)))` for the following class.

duck

```
(define node%
  (class object%
    (init [with-size : Real])
    (define current-size : Real with-size)
    (define/public (size) current-size)
    (super-new)))
```

Structural typing 2/3

- typed/racket infers type `(Class (size (-> Zero)))` for the following classes.

duck

```
(define empty%
  (class object%
    (define/public (size) 0)
    (super-new)))
(ann empty% (Class (size (-> Real)))) ;; more general

(define mt%
  (class object%
    (define/public (size) 0)
    (super-new)))
```

Structural typing 3/3

- Since classes have types, so do objects; we can write functions that take objects from all three classes
- None of these is a subclass of the other.

duck

```
(define (m [arg : (Object (size (-> Real)))]) : Real
  (send arg size))

(test (m (new empty%)) 0)
(test (m (new mt%)) 0)
(test (m (new node% [with-size 2])) 2)
```

Java Subclass example

p. 169

```
class A { String who = "A"; }
class B extends A { String who = "B"; }
class C extends A { String who = "C"; }
class D { String who = "D"; }
```

BB

```
System.out.println((true ? new B() : new B()).who);
```

BA

```
System.out.println((true ? new B() : new A()).who);
```

BC

```
System.out.println((true ? new B() : new C()).who);
```

BD

```
System.out.println((true ? new B() : new D()).who);
```

Typing `if`, again

p. 169

- `C ? T : E` is just Java notation for `(if C T E)`
- Recall our (`plait`-style) rule for `if`:

$$\frac{\Gamma \vdash \texttt{C} : \mathrm{Bool} \quad \Gamma \vdash \texttt{T} : W \quad \Gamma \vdash \texttt{E} : W}{\Gamma \vdash (\texttt{if C T E}) : W}$$

With union types

$$\frac{\Gamma \vdash \texttt{C} : \mathrm{Bool} \quad \Gamma \vdash \texttt{T} : W \quad \Gamma \vdash \texttt{E} : Z}{\Gamma \vdash (\texttt{if C T E}) : (\mathsf{U}\ W\ Z)}$$

- In Java

$$\frac{\Gamma \vdash \texttt{C} : \mathrm{Bool} \quad \Gamma \vdash \texttt{T} : W \quad \Gamma \vdash \texttt{E} : Z}{\Gamma \vdash (\texttt{if C T E}) : (\mathsf{lub}\ W\ Z)}$$

- where (lub W Z) is the least upper bound

Subtyping and substitution

- X is a subtype of Y (written $X <: Y$) if an X can safely be substituted for a Y.
- The simple cases are simple: an `Integer` can be substituted for a `Number`.
- Java says subclasses are subtypes, which means they only grow methods.
- With structural typing, we saw `(Object (size (-> Zero))` was a subtype of `(Object (size (-> Real))` which is similar, but more subtle.