CSE 341 Section 10

Subtyping, Review, and The Future

Outline

1. Subtyping
   - Overview
2. Review
   - Topics
   - Questions?
3. The Future
   - Languages
   - Courses

Records Overview

f* = field name
e* = expression
t* = type

Creation

```
{f0=e0, f1=e1, ..., fn=en}
```

Access/Update

```
e.field                    e1.field = e2
```

Type Signature

```
{f1:t1, f2:t2, ..., fn:tn}
```

Subtyping Overview

Subtyping Relation

`t1 <: t2` ≡ `t1` extends `t2` ≡ `t1` is a subtype `of t2`

Additional Type Rule

If `t1 <: t2` and `e` has type `t1`, then `e` also has type `t2`

Record Subtyping Rules

- Width subtyping: A supertype can have fewer fields
- Permutation subtyping: A supertype can have reordered fields
- Transitivity: If `t1 <: t2` and `t2 <: t3`, then `t1 <: t3`.
- Reflexivity: `t <: t` for any `t` (anything is a subtype of itself)

Function Types

Function Subtyping Rules

If `t2 <: t4` and `t3 <: t1`, then `t1 -> t2 <: t3 -> t4`.

or a more tangible example..

If `Cat <: Animal` and `Teacher <: Person`, then

`Person -> Cat <: Teacher -> Animal`.

- Function subtyping is covariant for their return types
- Function subtyping is contravariant for their argument types

- covariant: preserves subtype relation of types
- contravariant: reverses the subtype relation of types

Objects (in relation to records)

- Objects are basically the same as records except there is a distinction between mutable and immutable fields.
  - Mutable fields are instance variables
  - Immutable fields are methods
- Subtyping of objects happens almost the same way as records
  - e.g. Java/C# disallow contravariant method arguments
- The implicit self parameter in methods is **covariant** (unlike explicit arguments which are contravariant)

subclassing vs subtyping

- Java confuses these ideas as a matter of convenience, but you should keep these ideas separate
- Classes: define an object's behavior
- Types: describes what fields an object has and what messages it can respond to
- Subclassing: inherits behavior, modifies behavior via extension and overriding
- Subtyping: is a question of suitability and what we want to flag as a type error

Pop Quiz

Are these sound or not? (if not, give a counter-example)

- When overriding a method, we can change an argument type to be a supertype of what it was in the superclass' method.
  - Sound (contravariant argument types)
- When overriding a method, we can change an argument type to be a subtype of what it was in the superclass' method.
  - Unsound (covariant argument types)
- When overriding a method, we can change the result type to be a supertype of what it was in the superclass' method.
  - Unsound (contravariant return types)

Pop Quiz (continued)

Are these sound or not? (if not, give a counter-example)

- When overriding a method, we can change the result type to be a subtype of what it was in the superclass' method.
  - Sound (covariant return types)
- A subclass can change the type of a (mutable) field to be a subtype of what it was in the superclass. (This is changing the type of a field, not adding a second field.)
  - Unsound (depth subtyping on mutable fields)
- A subclass can change the type of a (mutable) field to be a supertype of what it was in the superclass. (This is changing the type of a field, not adding a second field.)
  - Unsound (depth subtyping on mutable fields)

At a Glance

- Benefits of no mutation
- Algebraic datatypes, pattern matching
- Higher-order functions; closures; tail recursion
- Lexical scope
- Currying; syntactic sugar
- Equivalence and side-effects
- Type inference
- Dynamic vs. static typing
- Laziness, streams, and memoization
- Macros
- Dynamic dispatch; double-dispatch
- Multiple inheritance, interfaces, and mixins
- OO vs. functional decomposition and extensibility
- Subtyping for records, functions, and objects
- Class-based subtyping
- Parametric polymorphism; bounded polymorphism

Questions?

What are your questions?

Some Exciting Developments…

- Rust (a “better” C / C++)
  - Type inference, higher-order functions
  - Concurrency “baked-in”
  - Eliminates null pointer exceptions
  - Improved memory management
- Scala (a “better” Java?)
  - FP + OOP + static typing + JVM
- Clojure (modern, concurrency-focused Lisp hosted on the JVM)
  - Persistent, immutable data structures
  - Concurrency primitives with an STM: atoms, vars, agents; refs
- Haskell (lazy, pure ML-like language)
  - Category theory: Monads, Monoids, Functors, . . .
  - Type classes, parsec, super-awesome type system, . . .
- And many more! Haxe, Groovy, Dart, Go, ecmascripten / asm.js, …

Future Courses

- CSE333 – Systems Programming
- CSE401 – Compilers
- CSE501 – Implementation of Programming Languages
- CSE505 – Concepts of Programming Languages