Lecture 20: Subtyping & OO Languages

CSC 131
Spring, 2019

Kim Bruce

Subtyping

- Can be added to non-OO languages
- Matching structures and signatures similar
  - but more restricted.
- Provides support for using values from new types in old (unexpected) contexts

Subtype Polymorphism

S is a *subtype* of T, written S <: T,

iff

a value of type S can be used in any context expecting a value of type T,

i.e., S can masquerade as a T.

Subsumption: e: S & S <: T ⇒ e: T.

Immutable Records

Records without field update (like Haskell/ML):

Sandwich = { bread: BreadType;
filling: FoodType }

s: Sandwich = { bread = rye;
filling = pastrami }

Only operation is extracting field:

... s.filling ...

Specializing Record Types

CheeseSandwich = {bread: BreadType;
filling: CheeseType;
sauce: SauceType}

c_s: CheeseSandwich = {bread = white;
filling = cheddar;
sauce = mustard}

Subtyping Immutable Records

If r: $\{ l_i : T_i \}_{1 \le i \le k}$ then expect r. $l_i : T_i$.

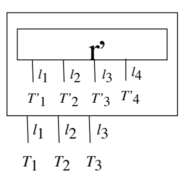
When is $\{ l_i : T'_i \}_{1 \le i \le n} <: \{ l_i : T_i \}_{1 \le i \le k}$?

Suppose r': $\{ l_i : T'_i \}_{1 \le i \le n}$

When can r' *masquerade* as elt of $\{ l_i : T_i \}_{1 \le i \le k}$?

Need $r'.l_i : T_i$.

Masquerading



$\{ l_i : T'_i \}_{1 \le i \le n} <: \{ l_i : T_i \}_{1 \le i \le k}$

iff

$k \le n$ and for all $1 \le i \le k$, $T'_i <: T_i$.

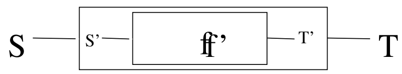
Functions

If $f : S \to T$ and $s : S$ then $f(s) : T$

When is $S' \to T' <: S \to T$?

If $f' : S' \to T'$, need $f'(s) : T$.

Subtyping Functions



$$S' \to T' <: S \to T$$
$$\text{iff}$$
$$S <: S' \text{ and } T' <: T.$$

Contravariant for parameter types.
Covariant for result types.

Variables

Variables can be *suppliers* & *receivers* of values.

x := x + 1

If x is a variable of type T, write x: ref T.

When is ref T' <: ref T?

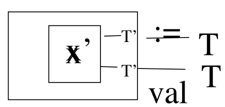
To replace variable x: ref T by x': ref T' in:

- expression: ... x ...

  Need T' <: T.
- assignment: x := e where e: T.

  Need T <: T'.

Variables



Supplier: covariant;
Receiver: contravariant

ref T' <: ref T iff T' ≈ T

Exercises

<u>Updatable Records</u>:

When is $\{ l_i : T'_i \}_{1 \le i \le n} <: \{ l_i : T_i \}_{1 \le i \le k}$?

... $r.l_i := e$ ...

More Exercises

Arrays:

- If S <: T, is Array of S <: Array of T?

Java says yes, but ...

not safe!

With few exceptions, for F: Types $\rightarrow$Types,
S <: T $\not\Rightarrow$ F(S) <: F(T).

Object-Oriented Languages

Roots in ADT Languages

- Ada and Modula-2 internal reps
  - couldn't be instantiated dynamically
  - no type or other method of organizing, despite similarity to records
  - provide better modules for building large systems
- Called object-based

Responding to the "SOFTWARE CRISIS!"

Qualities Desired in Software

- Correctness
- Robustness
- ~~Extensibility~~ *Almost all supported by ADT's*
- Reusability
- Compatibility

Object-Oriented Languages

- Objects that are data abstractions
- Objects have associated object type (classes or interfaces)
- Classes may *inherit* attributes from superclass
- Computations proceed by sending messages
- Subtype polymorphism
- Support dynamic method invocation

Programming Objects in ML

```
exception Empty;
fun newStack(x) =
   let
      val store = ref [x]
   in
      {push = fn y => store := y::(!store);
       pop = fn z => case !store of
                  nil = raise Empty
                | (y::ys) = (store := ys; y)
       }
   end;

val myStack = newStack(0);
#push(myStack)(1);
#pop(myStack)();
```

Parameter z ignored, used to delay evaluation really parameterless function

Weakness of ML

- No subtyping
- No this/self
- No inheritance
- *Similar issues in trying to do objects in LISP or other functional languages.*
- *Haskell doesn't have state!*

OO Keywords

- Object
- Message
- Class
- Instance
- Method
- Subtype
- Subclass

Objects

- Internal data abstractions
- Hide representation
- Have associated state
- Methods have access to its state
- Self

Object Types

- Allow objects to be first class
- Allow use in assignment, parameters, components of structures
- Allow objects to be classified via subtyping