Scribe: Helgi Ingolfsson, Shobhit Varshney Lecturer: Andrew Myers

1 Introduction

We would like to extend the traditionally typed λ-calculus for it to support objects, but objects are complicated beasts so we need to do this in a few steps. One of the key features of Object-oriented programming is the *subtype relationship*.

Subtyping was introduced in SIMULA, which started out as simulation packages in ALGOL. The designers of SIMULA introduced subtyping because they needed to implement algorithms that could be applied to several data types.

Here we will introduce a *subtype relationship* that can be used to extend our λ-calculus and show how it can be applied to functions and references. And a new record type. Here is a simple example of subtypes. Consider the figure 1 below:

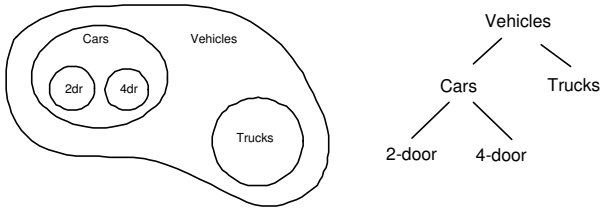

Figure 1: Subtyps as subsets example / Subtype hierarchy example

In this figure, Car and Truck are both subtypes of Vehicle, and 4-door and 2-door are both subtypes of Car. Which is defined as Car $\leq$ Vehicle, Truck $\leq$ Vehicle, 2-door $\leq$ Car, 4-door $\leq$ Car. Context expecting a Vehicle can also accept any of its subtypes.

2 Subtype relationships

Formally we notate the subtype relationship as: $\tau_1 \leq \tau_2$, which means τ_1 is a subtype of τ_2 or alternately, all values of type τ_1 are of type τ_2.

In the subset context $\tau_1 \leq \tau_2$ implies $[\![\tau_1]\!] \subseteq [\![\tau_2]\!]$, see figure 1 above.

2.1 Subtyping rules

We must carefully define our subtyping rules because they must preserve soundness. The basic rule for subtyping are:

$$\frac{\Gamma \vdash e : \tau \qquad \tau \leq \tau'}{\Gamma \vdash e : \tau'} \qquad \text{Subsumption}$$

Since the $\leq$ relation is reflexive and transitive, we have a *pre-order*.

$$\frac{}{\tau \leq \tau} \qquad \text{Reflexivity}$$

$$\frac{\tau_1 \leq \tau_2 \qquad \tau_2 \leq \tau_3}{\tau_1 \leq \tau_3} \qquad \text{Transitivity}$$

2.2 Type hierarchy

The subtype relationships for the 1 and 0 types are:

- $\overline{\tau \leq 1}$: All you can do with 1 is pass it along. If something is expecting type unit, it can accept any type, that is any type is as good as unit.
- $\overline{0 \leq \tau}$: 0 Can be accepted instead of any type because the context receiving this type never receives a value therfore it can not say that 0 is not good enough.

This can be represented graphically as:

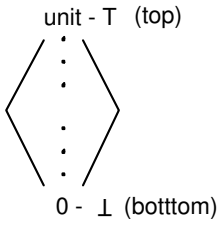

Figure 2: Type hierarchy

2.3 More Subtyping rules

$$\frac{\tau_1 \leq \tau_1' \qquad \tau_2 \leq \tau_2'}{\tau_1 \times \tau_2 \leq \tau_1' \times \tau_2'} \qquad \frac{\tau_1 \leq \tau_1' \qquad \tau_2 \leq \tau_2'}{\tau_1 + \tau_2 \leq \tau_1' + \tau_2'}$$

3 Function Subtyping

We would be tempted to write:

$$\frac{\tau_1 \leq \tau_1' \qquad \tau_2 \leq \tau_2'}{\tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'} \qquad \textbf{INCORRECT}$$

This would make our type system unsound! This is the typing rule that was implemented in the language Eiffel and because of it's incorrectness they needed to add runtime checking to make the language type save. What we relay want is to change the right hand side of the premise from *Covariance* to *Contravariance*. That is $\tau_1 \leq \tau_1'$ to $\tau_1' \leq \tau_1$. Giving us:

$$\frac{\tau_1' \leq \tau_1 \qquad \tau_2 \leq \tau_2'}{\tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'} \qquad \textbf{CORRECT}$$

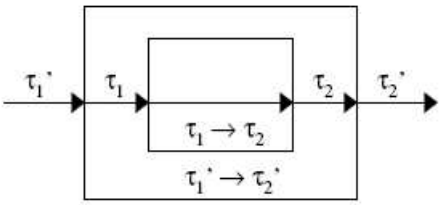

Figure 3: Function Subtyping

4 Record types

We extend our language to include record types.

$$\begin{array}{rcl} e & ::= & \cdots \mid \{x_1 = e_1,\ x_2 = e_2, \ldots,\ x_n = e_n\} \mid e.x \\ \tau & ::= & \cdots \mid \{x_1 : \tau_1,\ \ x_2 : \tau_2, \ldots,\ x_n : \tau_n\} \\ eval & ::= & \cdots \mid \{x_1 = v_1,\ x_2 = v_2, \ldots,\ x_n = v_n\}.x_i \to v_i \end{array}$$

We add the following derivation as well:

$$\frac{\Gamma \vdash e_i : \tau_i \quad ^{\forall i \in 1..n}}{\Gamma \vdash \{x_1 = e_1,\ x_2 = e_2, \ldots,\ x_n = e_n\} : \{x_1 : \tau_1,\ x_2 : \tau_2, \ldots,\ x_n : \tau_n\}}$$

Where $\Gamma \vdash e_i : \tau_i \quad ^{\forall i \in 1..n}$ is a short hand for writing all the n premises.

We consider two forms of subtyping:

- Depth subtyping means the fields of a subtype are subtypes themselves:

$$\frac{\tau_i \leq \tau_i' \quad ^{\forall i \in 1..n}}{\{x_1 : \tau_1,\ x_2 : \tau_2, \ldots,\ x_n : \tau_n\} \ \leq\ \{x_1 : \tau_1',\ x_2 : \tau_2', \ldots,\ x_n : \tau_n'\}}$$

- Width subtyping means a subtype has extra fields:

$$\frac{}{\{x_1 : \tau_1,\ \ x_2 : \tau_2, \ldots,\ x_n : \tau_n,\ x_{n+1} : \tau_{n+1}\} \ \leq\ \{x_1 : \tau_1,\ x_2 : \tau_2, \ldots,\ x_n : \tau_n\}}$$

Generalizing the Width subtyping for $n \leq m$ and merging the two rules to gather we have:

$$\frac{n \leq m \qquad \tau_i \leq \tau_i' \quad ^{\forall i \in 1..n}}{\{x_1 : \tau_1,\ x_2 : \tau_2, \ldots,\ x_m : \tau_m\} \ \leq\ \{x_1 : \tau_1',\ x_2 : \tau_2', \ldots,\ x_n : \tau_n'\}}$$

This rule implicitly says that the order of the fields does matter, and we can extend it only on the right hand side. This ordering makes it easy to computer addresses for fields at compile time. Some languages implement a version where the ordering does not matter, but that makes it computationally much more expensive.

5 References

We add references to our language be adding:

$$\begin{array}{rcl} e & ::= & \cdots \mid \text{ref } e \mid\ !e \mid e_1 := e_2 \\ \tau & ::= & \cdots \mid \tau \text{ ref} \end{array}$$

And the rules:

$$\frac{\Gamma \vdash\ e : \tau}{\Gamma \vdash\ \text{ref } e : \tau \text{ ref}} \qquad \frac{\Gamma \vdash\ e : \tau \text{ ref}}{\Gamma \vdash\ !e : \tau} \qquad \frac{\Gamma \vdash\ e_1 : \tau \text{ ref} \qquad \Gamma \vdash\ e_2 : \tau}{\Gamma \vdash\ e_1 := e_2 : 1}$$

Now for the subtyping rule, intuitively we would want something like:

$$\frac{\tau_1 \leq \tau_1'}{\tau \text{ ref} \leq \tau' \text{ ref}} \qquad \textbf{INCORRECT}$$

but this rule is not sound. Consider the following example, supposing that Car has a field trunk that Truck lacks: :

```
let x:Car ref = ref sedan in
   let y:Vehicle ref = x in
      y := largetrailer; !x.trunk
```

The most permissive correct rule is just the reflexive rule:

$$\frac{}{\tau \text{ ref} \leq \tau \text{ ref}} \qquad \text{that is subtyping is invariant}$$