CS 4110

Programming Languages & Logics

Lecture 26
Records and Subtyping

3 November 2014

Announcements

- Foster office hours 4-5pm
- Prelim II conflicts
- Next Thursday: Talk on *Iron* by Yaron Minsky PhD '02

Records

We have previously seen binary products (pairs of values), which can be generalized to *n*-ary products, also called *tuples*.

Records are a generalization of tuples

We annotate each field with a *label* drawn from a set $\mathcal{L}$

Example:

$$\{\mathrm{foo} = 32, \mathrm{bar} = \mathrm{true}\}$$

is a record value with an integer field foo and a boolean field bar.

The type of the record value is written

$$\{\mathrm{foo} : \mathbf{int}, \mathrm{bar} : \mathbf{bool}\}$$

Syntax

$$l \in \mathcal{L}$$

$$e ::= \cdots \mid \{l_1 = e_1, \ldots, l_n = e_n\} \mid e.l$$

$$v ::= \cdots \mid \{l_1 = v_1, \ldots, l_n = v_n\}$$

$$\tau ::= \cdots \mid \{l_1 : \tau_1, \ldots, l_n : \tau_n\}$$

Dynamic Semantics

$$
\begin{aligned}
E ::= {} & \ldots \\
& \mid \{l_1 = v_1, \ldots, l_{i-1} = v_{i-1}, l_i = E, l_{i+1} = e_{i+1}, \ldots, l_n = e_n\} \\
& \mid E.l
\end{aligned}
$$

$$
\frac{}{\{l_1 = v_1, \ldots, l_n = v_n\}.l_i \to v_i}
$$

Static Semantics

$$\frac{\forall i \in 1..n. \quad \Gamma \vdash e_i : \tau_i}{\Gamma \vdash \{l_1 = e_1, \ldots, l_n = e_n\} : \{l_1 : \tau_1, \ldots, l_n : \tau_n\}}$$

$$\frac{\Gamma \vdash e : \{l_1 : \tau_1, \ldots, l_n : \tau_n\}}{\Gamma \vdash e.l_i : \tau_i}$$

Example

Note that the order of labels is important!

The type of the record value

$$\{\mathrm{lat} = -40, \mathrm{long} = 175\}$$

is

$$\{\mathrm{lat} : \mathbf{int}, \mathrm{long} : \mathbf{int}\},$$

which is different from

$$\{\mathrm{long} : \mathbf{int}, \mathrm{lat} : \mathbf{int}\},$$

the type of the record value

$$\{\mathrm{long} = 175, \mathrm{lat} = -40\}$$

Subtyping

Definition (Subtype)

τ_1 is a subtype of τ_2 (written $\tau_1 \leq \tau_2$) if a program can use a value of type τ_1 whenever it would use a value of type τ_2.

If $\tau_1 \leq \tau_2$, then τ_1 is usually referred to as the subtype, and τ_2 as the supertype.

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

This typing rule says that if e has type τ and τ is a subtype of τ', then e also has type τ'.

Subtyping Relation

One can think of types as describing sets of values that share some common property. Then type τ is a subtype of type τ' is every value in the set for τ can be regarded as a value in the set for τ'.

The subtype relation is both reflexive and transitive. These properties are intuitive if we think of subtyping as a subset relation.

$$\frac{}{\tau \leq \tau}\ \text{S-Refl} \qquad\qquad \frac{\tau_1 \leq \tau_2 \quad \tau_2 \leq \tau_3}{\tau_1 \leq \tau_3}\ \text{S-Trans}$$

Record Subtyping

We certainly want "width" subtyping...

$$\frac{k \geq 0}{\{l_1 : \tau_1, \ldots, l_{n+k} : \tau_{n+k}\} \leq \{l_1 : \tau_1, \ldots, l_n : \tau_n\}}$$

...as well as "depth" subtyping...

$$\frac{\forall i \in 1..n. \quad \tau_i \leq \tau_i'}{\{l_1 : \tau_1, \ldots, l_n : \tau_n\} \leq \{l_1 : \tau_1, \ldots, l_n : \tau_n\}}$$

... and "permutation" subtyping:

$$\frac{\pi \text{ a permutation on } 1..n}{\{l_1 : \tau_1, \ldots, l_n : \tau_n\} \leq \{l_{\pi(1)} : \tau_{\pi(1)}, \ldots, l_{\pi(n)} : \tau_{\pi(n)}\}}$$

Record Subtyping

Putting all three forms of record subtyping together:

$$\frac{\forall i \in 1..n.\ \exists j \in 1..m. \quad l'_i = l_j \ \ \wedge \ \ \tau_j \leq \tau'_i}{\{l_1 : \tau_1, \ldots, l_m : \tau_m\} \leq \{l'_1 : \tau'_1, \ldots, l'_n : \tau'_n\}} \text{ S-Record}$$

Top Type

It is natural to ask... what is the maximal type with repsect to subtyping?

$$\tau ::= \cdots \mid \top$$

The $\top$ type can be used to model types such as Java's `Object`.

The subtyping rule for $\top$ is as follows:

$$\frac{}{\tau \leq \top}\ \text{S-Top}$$

Sum and Product Subtyping

We can extend the subtyping relation to handle sums and products, in the obvious way:

$$\frac{\tau_1 \leq \tau_1' \quad \tau_2 \leq \tau_2'}{\tau_1 + \tau_2 \leq \tau_1' + \tau_2'} \text{ S-Sum}$$

$$\frac{\tau_1 \leq \tau_1' \quad \tau_2 \leq \tau_2'}{\tau_1 \times \tau_2 \leq \tau_1' \times \tau_2'} \text{ S-Product}$$

Function Types

Consider two function types $\tau_1 \to \tau_2$ and $\tau_1' \to \tau_2'$.

What subtyping relations between the τ_i and τ_i' must hold to ensure that $\tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'$ holds?

> *"As usual, something funny happens to the left of the arrow"*
>
> *—John C. Reynolds*

Example

Consider the following expression,

$$G \triangleq \lambda f\!:\!\tau_1' \to \tau_2'.\ \lambda x\!:\!\tau_1'.\ f\,x.$$

which has type:

$$(\tau_1' \to \tau_2') \to \tau_1' \to \tau_2'$$

Now suppose we want to supply $h : \tau_1 \to \tau_2$ to G

Suppose that v is a value of type τ_1'. Then $G\ h\ v$ will evaluate to $h\ v$, meaning that h will be passed a value of type τ_1. Since h has type $\tau_1 \to \tau_2$, we must have $\tau_1' \leq \tau_1$

The result type of $G\ h\ v$ should be of type τ_2' according to the type of G, but $h\ v$ will produce a value of type τ_2, as indicated by the type of h. So we must have $\tau_2 \leq \tau_2'$

Function Subtyping

Putting these two pieces together, we get the following subtyping rule for function types:

$$\frac{\tau_1' \leq \tau_1 \quad \tau_2 \leq \tau_2'}{\tau_1 \rightarrow \tau_2 \leq \tau_1' \rightarrow \tau_2'} \text{ S-Function}$$

Note that the subtyping relation between the argument and result types in the premise are in different directions!

The subtype relation for the result type is in the same direction as for the conclusion (primed version is the supertype, non-primed version is the subtype); it is in the opposite direction for the argument type.

We say that subtyping for the function type is *covariant* in the result type, and *contravariant* in the argument type.

Reference Subtyping

Suppose we have a location *l* of type τ **ref**, and a location *l*′ of type τ' **ref**.

What should the relationship be between τ and τ' in order to have τ **ref** $\leq$ τ' **ref**?

Example

Consider the following program R, which takes a location x of type τ' **ref** and reads from it.

$$R \triangleq \lambda x\!:\!\tau'\ \textbf{ref}.\ \ !x$$

This has the type τ' **ref** $\to \tau'$. Suppose we give R a location l as an argument. Then $R\ l$ will look up the value stored in l, and return a result of type τ (since l is type τ **ref**.

Since R is meant to return a result of type τ' **ref**, we thus want to have $\tau \leq \tau'$.

Example

Now consider the following program W, which takes a location x of type τ' **ref**, a value y of type τ', and writes y to the location.

$$W \triangleq \lambda x : \tau' \ \mathbf{ref}.\ \lambda y : \tau'.\ x := y$$

This program has type τ' **ref** $\rightarrow \tau' \rightarrow \tau'$.

Suppose we have a value v of type τ', and consider the expression $W\ l\ v$.

This will evaluate to $l := v$, and since l has type τ **ref**, it must be the case that v has type τ, and so $\tau' \leq \tau$.

This suggests that subtyping for reference types is contravariant!

Reference Subtyping

In fact, subtyping for reference types must be *invariant*: a reference type τ **ref** is a subtype of τ' **ref** if and only if $\tau \leq \tau'$ and $\tau' \leq \tau$.

$$\frac{\tau \leq \tau' \quad \tau' \leq \tau}{\tau\,\textbf{ref} \leq \tau'\,\textbf{ref}}\ \text{S-Ref}$$

Indeed, it is not hard to see that to be sound, subtyping for all mutable language constructs must be invariant

Java Arrays

Interestingly, in the Java programming language, mutable arrays have covariant subtyping!

Suppose that we have two classes Person and Student such that Student extends Person (that is, Student is a subtype of Person).

Code that only reads from arrays typechecks,

$$\text{Person[ ] arr} = \text{new Student[ ] \{ new Student(“Alice”) \};}$$
$$\text{Person p} = \text{arr[0];}$$

but the following code, which writes into the array, has some issues:

$$\text{arr[0]} = \text{new Person(“Bob”);}$$

Specifically, this generates an ArrayStoreException