Do it yourself: Design and metatheory of a λ-calculus with subtyping

2018-12-14

The objectives of this session:

- Design a type system from scratch.
- Adapt the proof techniques already seen during the first semester.
- Introduce subtyping.

Instructions:

- Do the exercise at your own pace, ask for help from me or your classmates. It is more important to do things properly than to do them quickly.
- You can send me your answers by email [1] or by dropping them in my mail box at the third floor of Sophie Germain building. If I receive them before next Tuesday, I will give you my feedback during the next session.

1 Syntax and Semantics

Let us consider the following syntax for terms of a λ-calculus with records, that we will call $\lambda_{<:}$:

$$
\begin{array}{rcl}
t & ::= & x \\
& | & 0, 1, \ldots \\
& | & t\,u \\
& | & \lambda(x : \tau).t \\
& | & \{\ell_1 = t_1; \ldots; \ell_n = t_n\} \\
& | & t.\ell
\end{array}
\qquad
\begin{array}{rcl}
\tau & ::= & \top \\
& | & \mathsf{nat} \\
& | & \tau \to \tau \\
& | & \{\ell_1 : \tau_1; \ldots; \ell_n : \tau_n\}
\end{array}
$$

where $x, y, \ldots$ denote identifiers taken in some enumerable set $\mathcal{I}$ and where $\ell, \ell_1, \ell_2, \ldots$ denote labels taken in some enumerable set $\mathcal{L}$. We want to equip this language with a call-by-value weak reduction strategy.

Exercise 1:

1. Define the syntax for values v.
2. Define a syntax for contexts C that encode the reduction strategy we are targetting.
3. Give the reduction rules.
4. Prove that for every reducible term t, there is a unique decomposition of the form $t = C[u]$ so that u is a redex.

[1] `yrg@irif.fr`

2 Declarative type system

Consider the following function:

$$\lambda(r : \{\ell : \tau\}).r.\ell$$

, it can be safely applied to every record that contains at least a field ℓ of type τ. In simply-typed λ-calculus, the valid arguments for this function are the records that contain exactly one field ℓ of type τ. Hence, STLC is strict restriction of what can be safely allowed. The purpose of the subtyping relation is to remove this restriction.

We introduce a relation written "$\tau <: \tau'$" which is read "τ is a subtype of τ'. Intuitively, if $\tau <: \tau'$ holds, then every term of type τ can be used where a term of type τ' is valid. This is formally captured by the following (Subsomption) rule:

$$\text{Subsomption}\quad \frac{\Gamma \vdash t : \tau \qquad \tau <: \tau'}{\Gamma \vdash t : \tau'}$$

The declarative type system of our calculus is defined as the extension of simply-typed λ-calculus by the (Subsomption) rule and two typing rules for record creation (Record) and field access (Field).

Intuitively, the subtyping relation should be a preorder, that is, it should be reflexive and transitive, since it accounts for a notion of valid subsomption. $\top$ can be seen as a non informative type

It must also witness the informal fact that "the extension of a record with more fields is at least as capable as its non extended counterpart". This relation must not be sensitive to the order of appearance of the fields in the record types since they do not have an influence on the presence or absence of a given field.

Exercise 2:

1. Give the two typing rules (Record) and (Field).
2. Assume succ : nat $\to$ nat. Consider the function

   $$(\lambda(f :?).\text{succ}((f\{\ell_1 = 31; \ell_2 = 73\}).\ell_3))$$

   , can you characterize the types that make valid ascriptions for f?
3. Propose a set of rules to define the subtyping relation $\tau <: \tau'$.
4. Is the resulting relation a partial order?
5. State and prove the "Subject Reduction" property. It is recommended to decompose the proof through the introduction of the usual Lemmas about inversions of judgements and substitutions.
6. State and prove the "Progress" property.

3 Algorithmic type system

You may have noticed that the declarative type system is not syntax-directed: the subsomption rule can apply at any place in the typing derivation. Fortunately, one can prove that the declarative type system is equivalent [2] to an algorithmic one that is obtained by removing the rule (App) for applications and the rule (Subsomption) by the following single rule (App-Sub):

$$\frac{\Gamma \vdash t : \tau_1' \to \tau_2 \qquad \Gamma \vdash u : \tau_1 \qquad \tau_1 <: \tau_1'}{\Gamma \vdash t\, u : \tau_2}$$

[2] For a notion of equivalence that must be properly defined.

Exercise 3:

1. Given two types τ_1 and τ_2, can you find an algorithm that decides if $\tau_1 <: \tau_2$ holds? Prove that it indeed terminates and that it is correct.

2. Consider a typing derivation whose conclusion is obtained by an application of the rule (Abs) immediately preceded by an application of the (Subsomption) rule. Find a typing derivation for the same judgment but where (Subsomption) rule has been pushed down in the typing derivation, i.e. find a way to commute (Abs) and (Subsomption) in the previous typing derivation.

3. Consider now the case for a typing derivation whose conclusion is obtained by an application of rule (App) immediately preceded by an application of the (Subsomption) rule on the left-hand-side hypothesis. Is it possible to find a typing derivation where the (Subsomption) is pushed at the bottom? Same question if the (Subsomption) rule is located on the right-hand side.

4. Study the case for two successive applications of the rule (Subsomption) and the cases for the rules (Record) and (Field).

5. By analogy with the proof of equivalence between the declarative and algorithmic type system of ML, introduce two explicitly-typed languages $e\lambda_{<:}$ and $x\lambda_{<:}$ for $\lambda_{<:}$ whose type erasures match $\lambda_{<:}$. The language $e\lambda_{<:}$ is equipped with the same rules as the declarative type system of $\lambda_{<:}$. The language $x\lambda_{<:}$ is equipped with the algorithmic typing rules. Define a normalization procedure from the typing derivations of $e\lambda_{<:}$ to the typing derivations of $x\lambda_{<:}$ whose judgment is written:
$$\Gamma \vdash t \in e\lambda_{<:} : \tau \Rightarrow t' \in x\lambda_{<:} : \tau'$$

6. Assume that $\Gamma \vdash t \in e\lambda_{<:} : \tau \to t' \in x\lambda_{<:}$. The assertion "If $\Gamma \vdash t' : \tau$ holds in $x\lambda_{<:}$, then $\Gamma \vdash t : \tau$ holds in $e\lambda_{<:}$. Can you explain why? Find a valid relation between the type assigned to a program by the algorithmic type system of $x\lambda_{<:}$ and the types of this program in the declarative type system of $e\lambda_{<:}$.

4 Going further

Exercise 4:

1. An extension of $\lambda_{<:}$ with references (mutable cells) requires the introduction of a type **ref** τ. How would you write the subtyping rule for these reference types? Justify your answer.

2. Imagine that we introduce $\bot$, a dual to $\top$, which is a subtype of any type, i.e. $\bot <: \tau$. Why must this type be empty? (i.e. It cannot be inhabited to preserve type satefy.) What would be its usage, then?