Do it yourself:
Design and metatheory
of a λ-calculus with dependent types.

2018-01-12

The objectives of this session:

- Design a type system from scratch.
- Introduce a dependently-typed representational language.
- Study the implementation techniques for conversion rules.

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 Type system

We consider the following syntax for λ_π, a λ-calculus with dependent types.

$$
\begin{array}{rcll}
t & ::= & x & \text{Terms} \\
 & | & t\,u & \\
 & | & \lambda(x:\tau).t &
\end{array}
\qquad
\begin{array}{rcll}
\tau & ::= & \alpha & \text{Types} \\
 & | & \pi(x:\tau).\tau & \\
 & | & \tau\,t &
\end{array}
$$

where $x, y, \ldots$ denote identifiers taken in some enumerable set $\mathcal{I}$ and $\alpha, \beta, \ldots$ denote type identifiers taken in some enumerable set $\mathcal{V}$.

One specificity of this type algebra is to include a **dependent product** construction for function types. The inhabitants of the type $\pi(x : \tau_1).\tau_2$ are the functions that take a value x of type τ_1 as input to produce a value of type τ_2 where x can be a free variable of τ_2. Typically, the constructor for vectors of T can be given the types:

$$
\begin{array}{rcl}
\text{nil} & : & \text{list}\ 0 \\
\text{cons} & : & \pi(n : \textbf{nat}).\pi(x : T).\pi(l : \text{list}\ n).\text{list}\ (n+1)
\end{array}
$$

For the sake of conciseness, we write $\tau_1 \rightarrow \tau_2$ for $\pi(x : \tau_1).\tau_2$ where x is not free in τ_2. For instance:

$$\text{cons} : \pi(n : \textbf{nat}).T \rightarrow \text{list}\ n \rightarrow \text{list}\ (n+1)$$

[1] `yrg@irif.fr`

Exercise 1:

1. Notice that terms can appear in types. Therefore, a type can be ill-formed because it contains an ill-typed term. Propose a syntax for kinds, which will serve as "types for types". Deduce the shape of the judgments needed to define the type system of λ_π.

2. Propose typing rules for term variables, term abstractions and term applications.

3. Motivate the introduction of a conversion rule for this type system. What equivalence should we use on types? on kinds? Give the rules for these equivalences.

2 λ_π as a representational language

Consider the following (incomplete) signature:

$$
\begin{array}{rcl}
\text{term} & : & \star \\
\text{app} & : & \text{term} \to \text{term} \to \text{term} \\
\text{lam} & : & \text{ty} \to (\text{term} \to \text{term}) \to \text{term} \\
\text{ty} & : & \star \\
\text{arrow} & : & \text{ty} \to \text{ty} \to \text{ty} \\
\text{iota} & : & \text{ty} \\
\text{of} & : & \text{term} \to \text{ty} \to \star \\
\text{tc-app} & : & \pi(tu : \text{term}).\pi(ab : \text{ty}).\text{of}\, t\, (\text{arrow}\, a\, b) \to \text{of}\, u\, a \to \text{of}\, (\text{app}\, t\, u)\, b \\
\text{tc-lam} & : & ?
\end{array}
$$

This signature is meant to encode the simply typed λ-calculus using the technique called higher-order abstract syntax. Roughly speaking, the idea behind this encoding is to reuse the λ-abstraction of λ_π to encode the λ-abstraction of the object language (here the STLC). For instance, the term of $\lambda(x : \iota).x$ is encoded by:

$$\text{lam iota}\ (\lambda x.x)$$

1. Why is it no constant for the variable constructor of the STLC terms?

2. Propose a type for tc-lam.

3. Give the well-typed term that encodes the following judgment of STLC:

$$\emptyset \vdash (\lambda(f : \iota \to \iota)(x : \iota).fx)(\lambda(x : \iota).x) : \iota \to \iota$$

4. Why is λ_π often called a "Logical Framework"?

3 Efficient convertibility decision procedure

1. Is every well-typed term of λ_π strongly normalizing?

2. Deduce a naive algorithm from the previous question.

3. What are weak head normal forms?

4. How weak head normalization can help in implementing a more efficient decision procedure (in the negative case)?