Simply Typed Lambda Calculus

Pramook Khungurn

December 11, 2019

This handout follows Pierce's treatment of simply typed lambda calculus. We'll go over simpler typed lambda calculus over the type Bool and its normalization property.

1 Typing Relation

- There are two types of objects we are dealing with.
  - The first are **terms**, which are lambda terms.
  - The second are **types**, which can be thought of as sets of terms with certain properties.
- A **typing relation** is the relation $t : T$, where t is a term and T is a type.
  It means t *obviously* evaluates to a value of an appropriate form.
  We should be able to *statically* see this relation without doing any evaluation of t.
- Most typing relations are *conservatitve*. They can only make use of dynamic information.
  For example, we would not be able to determine type of `if true then 0 else false` although the evalution gives you only one possible value.
- A typing relation is defined as the smallest binary relation that satisfies a number of inference rules assigning types to terms.
- A term t is **typable** (or **well typed**) if there is some type T such that $t : T$.
- We want our type system to be **safe**. This means they have the following two properties:
  - **Progress**. A well-typed term is not stuck.
    That is, it is either a value, or a term which has an evaluation rule it can take to proceed.
  - **Preservation**. If a well-typed term takes an evaluation step, the resulting term is also well typed.

2 Simple Types over Bool

- The set of **simple types** over Bool is generated by the following rules:
  - Bool itself is a simple type over Bool.
  - If T_1 and T_2 are simple types over Bool, then $T_1 \to T_2$ is a simple types over Bool.
- The type constructor $\to$ is right-associative.
  The expression $T_1 \to T_2 \to T_3$ means $T_1 \to (T_2 \to T_3)$.

- The interpretation is that $T_1 \to T_2$ is the type of functions that takes in input of type T_1 and maps it to an output of type T_2.

  For example, Bool $\to$ Bool is the function that takes a boolean and spits out a boolean. Let us call this type the type of boolean function.

  (Bool $\to$ Bool) $\to$ Bool $\to$ Bool is a funtion that takes a boolean function and spits out another boolean function.

- We will annotate the argument to the lambda term with the type it's supposed to have. For example, we will write $\lambda x : T.\ t$ instead of just $\lambda x.\ t$.

- Languages in which type annotations are used to help guide the typechecker are called **explicitly typed**. Languages in which we ask the typechecker to infer this information is called **implicitly typed**.

- When typing an abstraction $\lambda x : T_1.\ t_2$, the type of the abstraction is the type T_2 that you get when you assume that all the occurence of type x in t_2 is of type T_1.

  This is captured by the following inference rule:

$$\frac{x : T_1 \vdash t_2 : T_2}{\vdash \lambda x : T_1.\ t_2 : T_1 \to T_2}$$

- The notation $\Gamma \vdash t : T$ is used to denote a three-place relation where (1) Γ is a set of assumptions, (2) t is a term, and (3) T is a type. It means that if Γ holds, then t has type T.

  When the set of assumptions is the empty set, we would just right $\vdash t : T$.

- Most of the times, Γ will have the form $x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n$. So we can think of Γ as a function that maps variables to their types. As such, we can use $\mathrm{dom}(\Gamma)$ to denote the variables bound by Γ.

- The rule of typing abstraction has the following general form:

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma, \vdash \lambda x : T_1.\ t_2 : T_1 \to T_2}$$

- A variable has the type that we assume it to have:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}$$

- Here is the typing rule for application:

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash (t_1\ t_2) : T_{12}}$$

- Lastly, here is the typing rule for the if statement:

$$\frac{\Gamma \vdash t_1 : \mathrm{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash (\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3) : T}$$

3 Properties of Typing