Logical relations: safety of simply-typed lambda calculus

Amin Timany

August 16, 2016

Note: in these notes, we simply ignore the issues regarding the clash between variable names, e.g., capturing, by assuming that bound variables are renamed whenever necessary to avoid such problems.

1 Language

1.1 Syntax

$$\begin{array}{lll} \textit{variables}(\mathsf{Var}) & x, y, z, \ldots & \\ \textit{expressions}(\mathsf{E}) & e \quad ::= & x \mid tt \mid (e, e) \mid \mathit{fst}\ e \mid \mathit{snd}\ e \mid \lambda x.\ e \mid e\ e \\ \textit{values}(\mathsf{Val}) & v \quad ::= & tt \mid (v, v) \mid \lambda x.\ e \end{array}$$

The set of values is a subset of the set of expressions: $\mathsf{Val} \subset \mathsf{E}$.

1.2 Types and typing

$$\textit{types}(\mathsf{Typ}) \quad \tau \quad ::= \quad () \mid \tau \times \tau \mid \tau \to \tau$$

We consider typing contexts as unordered sequences associating variables to types.

$$\textit{typing context}(\mathsf{TCtx}) \quad \Gamma \quad ::= \quad \cdot \mid x : \tau, \Gamma$$

Typing rules for our simply-typed lambda calculus(STLC) are:

$$\frac{}{\Gamma \vdash tt : ()} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathit{fst}\ e : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathit{snd}\ e : \tau_2} \qquad \frac{x : \tau_1, \Gamma \vdash e : \tau_2}{\Gamma \vdash \lambda x.\ e : \tau_1 \to \tau_2}$$

we write $e : \tau$ as a shorthand for $\cdot \vdash e : \tau$.

1.3 Operational semantics (CBV)

We describe the small-step call-by-value (CBV) operational semantics for STLC. We do this in a way that in two steps. This is more-or-less the standard for describing the semantics of a CBV language. In the first step we give the head reduction relation ($\rightsquigarrow$). In the second step we extend this to non-head reductions using evaluation context (ECtx).

Head reduction:

$$\mathit{fst}\ (v_1, v_2) \rightsquigarrow v_1 \qquad \mathit{snd}\ (v_1, v_2) \rightsquigarrow v_2 \qquad (\lambda x.\ e)\ v \rightsquigarrow e[v/x]$$

Note that here v's are values *and not any expression*. $e[v/x]$ is the expression e where all instances of x are replaced with v. *Remember that all substitutions are capture avoiding*.

Non-head reduction: If the redex (what is being reduced) is not in the head position (see above) then evaluation contexts determine where in the term a reduction can happen.

$$\frac{e \rightsquigarrow e'}{\mathbf{K}[e] \rightsquigarrow \mathbf{K}[e']}$$

where $\mathbf{K}$ is an evaluation context, i.e., $\mathbf{K} \in \mathsf{ECtx}$ and $\mathbf{K}[e]$ is the expression where the single whole in the context $\mathbf{K}$ (see below) is substituted with e.

Evaluation Contexts:

$$\textit{evaluation contexts}(\mathsf{ECtx}) \quad \mathbf{K} \quad ::= \quad [\cdot] \mid \mathit{fst}\ \mathbf{K} \mid \mathit{snd}\ \mathbf{K} \mid (\mathbf{K}, e) \mid (v, \mathbf{K}) \mid \mathbf{K}\ e \mid v\ \mathbf{K}$$

Example: The following is the only possible reduction for the expression:

$$\mathit{fst}\ ((\lambda x.\ ((\lambda y.\ tt)\ x, (\lambda y.\ x)\ tt))\ tt)$$

$$\mathit{fst}\ ((\lambda x.\ ((\lambda y.\ tt)\ x, (\lambda y.\ x)\ tt))\ tt) \rightsquigarrow \mathit{fst}\ ((\lambda y.\ tt)\ tt, (\lambda y.\ tt)\ tt) \rightsquigarrow \mathit{fst}\ (tt, (\lambda y.\ tt)\ tt)$$
$$\rightsquigarrow \mathit{fst}\ (tt, tt) \rightsquigarrow tt$$

Exercise: Determine the evaluation context for each step of the reduction above.

We use $\rightsquigarrow^*$ to denote reflexive and transitive closure of $\rightsquigarrow$.

2 Type safety

We say a language is type safe or has the type safety property if:

$$\forall e, \tau.\ e : \tau \Rightarrow \forall e'.\ e \rightsquigarrow^* e' \Rightarrow e' \in \mathsf{Val} \lor \exists e''.\ e \rightsquigarrow e''$$

For such a simple language as STLC, type safety can be easily proven using the well-known progress and preservation technique. In these notes we ignore this well-known technique for the sake of developing one of the (if not the) simplest use cases of logical relations.

3 Logical relations