Harvard School of Engineering and Applied Sciences — CS 252: Advanced Topics in Programming Languages

Logical Relations Part 2

Lecture 2 Wednesday, September 6, 2017

In this lecture, we will continue exploring logical relations, by considering a proof that requires reasoning about *contextual equivalence*. The proof is enabled by using a logical relation over pairs of expressions that implies contextual equivalence of a pair, instead of directly reasoning about contextual equivalence. This lecture is based on lectures by Prof Amal Ahmed at the Oregon Programming Languages Summer School, 2015[1], as reported in the notes by Lau Skorstengaard.[2] This lecture and the next two follow the notes quite closely.

1 System F

We briefly recap System F (i.e., simply-typed lambda calculus with polymorphic types). For simplicity, our only base type is integers, and we don't have any operations over integers.

$$\begin{aligned} e &::= n \mid x \mid \lambda x\!:\!\tau.\, e \mid e_1\; e_2 \mid \Lambda X.\, e \mid e\;[\tau] \\ v &::= n \mid \lambda x\!:\!\tau.\, e \mid \Lambda X.\, e \\ \tau &::= \textbf{int} \mid \tau_1 \to \tau_2 \mid X \mid \forall X.\, \tau \end{aligned}$$

$$E ::= [\cdot] \mid E\; e \mid v\; E \mid E\;[\tau]$$

$$\frac{e \longrightarrow e'}{E[e] \longrightarrow E[e']} \qquad \frac{}{(\lambda x\!:\!\tau.\, e)\; v \longrightarrow e\{v/x\}} \qquad \frac{}{(\Lambda X.\, e)\;[\tau] \longrightarrow e\{\tau/X\}}$$

$$\frac{}{\Delta, \Gamma \vdash n\!:\!\textbf{int}} \qquad \frac{\Delta \vdash \tau\ \mathsf{ok}}{\Delta, \Gamma \vdash x\!:\!\tau}\ \Gamma(x) = \tau \qquad \frac{\Delta, \Gamma, x\!:\!\tau \vdash e\!:\!\tau' \quad \Delta \vdash \tau\ \mathsf{ok}}{\Delta, \Gamma \vdash \lambda x\!:\!\tau.\, e\!:\!\tau \to \tau'}$$

$$\frac{\Delta, \Gamma \vdash e_1\!:\!\tau \to \tau' \quad \Delta, \Gamma \vdash e_2\!:\!\tau}{\Delta, \Gamma \vdash e_1\; e_2\!:\!\tau'} \qquad \frac{\Delta \cup \{X\}, \Gamma \vdash e\!:\!\tau}{\Delta, \Gamma \vdash \Lambda X.\, e\!:\!\forall X.\, \tau} \qquad \frac{\Delta, \Gamma \vdash e\!:\!\forall X.\, \tau' \quad \Delta \vdash \tau\ \mathsf{ok}}{\Delta, \Gamma \vdash e\;[\tau]\!:\!\tau'\{\tau/X\}}$$

$$\frac{}{\Delta \vdash X\ \mathsf{ok}}\ X \in \Delta \qquad \frac{}{\Delta \vdash \textbf{int}\ \mathsf{ok}} \qquad \frac{\Delta \vdash \tau_1\ \mathsf{ok} \quad \Delta \vdash \tau_2\ \mathsf{ok}}{\Delta \vdash \tau_1 \to \tau_2\ \mathsf{ok}} \qquad \frac{\Delta \cup \{X\} \vdash \tau\ \mathsf{ok}}{\Delta \vdash \forall X.\, \tau\ \mathsf{ok}}$$

2 Theorems for Free and Contextual Equivalence

In System F, polymorphic functions are unable to inspect values of polymorphic type. This means that there are limitations on what polymorphic functions can do: their behavior has to be independent of the type of the values they are given. This parametricity gives us "theorems for free"[3]. For example, the following theorems hold.

Theorem 1. *If $\vdash e\!:\!\forall X.\, X$ and $\vdash \tau$ ok and $\vdash v\!:\!\tau$ then $e\;[\tau]\; v \longrightarrow^* v$ (i.e., e is the identify function).*

[1]Videos available at `https://www.cs.uoregon.edu/research/summerschool/summer15/curriculum.html`.
[2]Available at `https://www.cs.uoregon.edu/research/summerschool/summer16/notes/AhmedLR.pdf`.
[3]"Theorems for free!", by Philip Wadler, in *4th International Conference on Functional Programming and Computer Architecture*, 1989.

Theorem 2. *If $\vdash e : \forall X.\, X \to \textbf{int}$ and $\vdash \tau$ ok and $\vdash \tau'$ ok and $\vdash v : \tau$ and $\vdash v' : \tau'$ then $e\ [\tau]\ v$ and $e\ [\tau']\ v'$ cannot be distinguished by any program (i.e., $e\ [\tau]\ v$ and $e\ [\tau']\ v'$ are contextually equivalent).*

Let's formally define contextual equivalence. We first define *contexts*. Unlike evaluation contexts (which we've seen previously), these contexts allow a "hole" for any subexpression.

$$C ::= [\cdot] \mid C\ e \mid e\ C \mid C\ [\tau] \mid \lambda x : \tau.\, C \mid \Lambda X.\, C$$

We also need a notion of *context typing*. We write $C : (\Delta, \Gamma \vdash \tau) \Longrightarrow (\Delta', \Gamma' \vdash \tau')$ if context C has a hole that can be filled with an expression e such that $\Delta, \Gamma \vdash e : \tau$, and then $C[e]$ will have type τ' under context Δ', Γ', i.e., $\Delta', \Gamma' \vdash C[e] : \tau'$.

$$\frac{\Delta, \Gamma \vdash e : \tau \qquad \Delta', \Gamma' \vdash C[e] : \tau'}{C : (\Delta, \Gamma \vdash \tau) \Longrightarrow (\Delta', \Gamma' \vdash \tau')}$$

We can now define contextual equivalence.

Definition 1. Expressions e_1 and e_2 are *contextually equivalent*, written $\Delta, \Gamma \vdash e_1 \approx^{ctx} e_2 : \tau$, iff $\Delta, \Gamma \vdash e_1 : \tau$ and $\Delta, \Gamma \vdash e_2 : \tau$ and for all contexts C such that $C : (\Delta, \Gamma \vdash \tau) \Longrightarrow (\vdash \tau')$ we have

$$C[e_1] \longrightarrow^* v \iff C[e_2] \longrightarrow^* v$$

Intuitively, if two expressions are contextually equivalent, then (e_1 and e_2 are well-typed and) no context can distinguish the two expressions, i.e., can evaluate to different values depending on whether the context's hole is filled with e_1 or e_2.

Contextual equivalence allows us to talk about two programs being indistinguishable (by a context). This is a useful concept in many ways. It can allow us to talk about different implementations behaving the same. For example, given two implementations of a stack (say, one using a list, the other using an array), if the two implementations are contextually equivalent, then from the perspective of a client, they behave exactly the same. Or consider two programs that are identical except for a secret value (e.g., a password): if the two programs are contextually equivalent, it means that no context can learn anything about the secret value. Or consider an optimized version of a program: if the optimized version and the original program are contextually equivalent, then the optimizations did not modify the behavior of the program in any observable way.

So, how do we prove these theorems? We will define a logical relation such that if two expressions are in the relation, then they are contextually equivalent. That means we can prove our free theorems using the logical relation rather than contextual equivalence directly, but get the same result. If we tried to deal with contextual equivalence directly, we need to quantify over all possible program contexts, which turns out to be difficult.

3 Logical Relation for Contextual Equivalence

We will define two (families of) relations: one on values and one on expressions. Both relations are indexed by type. We will then use these two relations to define our logical relation that will imply contextual equivalence.

We will consider a few versions of these relations, until we get the right one.

Let's start with the value relation. Relation $\mathcal{V}_\tau$ will relate pairs of closed, well-typed values of type τ. (For brevity, we won't write this in our definitions.) Relate $\mathcal{E}_\tau$ will relate pairs of closed, well-typed expressions of type τ.

Let's have an initial attempt at defining the value relation.

$$\begin{aligned}\mathcal{V}_{\textbf{int}} &= \{(n, n) \mid n \in \mathbb{Z}\} \\ \mathcal{V}_{\tau \to \tau'} &= \{(\lambda x : \tau.\, e_1, \lambda x : \tau.\, e_2) \mid \forall (v_1, v_2) \in \mathcal{V}_\tau.\ (e_1\{v_1/x\}, e_2\{v_2/x\}) \in \mathcal{E}_{\tau'}\}\end{aligned}$$

Note that the definition for function types takes related inputs to related outputs.

We next consider polymorphic types $\forall X.\ \tau$, which have values of the form $\Lambda X.\, e$. We can use the same principle of taking related inputs to related outputs. Here, the inputs are types (i.e., we apply values $\Lambda X.\, e$ to types), and we don't actually need to have any notion of "related types", so we will allow any pair of types.(Note that we need an arbitrary pair of types, since we want to use this to prove Theorem 2, which allows arbitrary choice of types τ and τ'.) But what is the relation we should use for the outputs?

$$\mathcal{V}_{\forall X.\ \tau} = \{(\Lambda X.\, e_1, \Lambda X.\, e_2) \mid \forall \tau_1, \tau_2.\ (e_1\{\tau_1/X\}, e_2\{\tau_2/X\}) \in \mathcal{E}_{\tau\{???/X\}}\}$$

What is the relation we should use for the output? Clearly it should be something related to τ since e_1 and e_2 have type τ. But if we replace type variable X with either τ_1 or τ_2 (i.e., use either $\mathcal{E}_{\tau\{\tau_1/X\}}$ or $\mathcal{E}_{\tau\{\tau_2/X\}}$), then that would break the well-typedness requirements (since, e.g., $e_2\{\tau_2/X\}$ may not have type $\tau\{\tau_1/X\}$).

What we really need to do is replace type variable X with the pair of types (τ_1, τ_2)! We do this by parameterizing the entire logical relation with a *relational substitution*, i.e., a substitution ρ that maps type variables to pairs of types.

$$\mathcal{V}^{\rho}_{\forall X.\ \tau} = \{(\Lambda X.\, e_1, \Lambda X.\, e_2) \mid \forall \tau_1, \tau_2.\ (e_1\{\tau_1/X\}, e_2\{\tau_2/X\}) \in \mathcal{E}^{\rho[X \mapsto (\tau_1, \tau_2)]}_{\tau}\}$$

This substitution ρ also gives us a way to define the relation for type variables! Well, almost...

$$\mathcal{V}^{\rho}_{X} = \{(v_1, v_2) \mid \rho(X) = (\tau_1, \tau_2) \text{ and } (v_1, v_2) \in \mathcal{V}_{???}\}$$

What relation should we use to restrict the pair of values (v_1, v_2)? We again face a similar problem, that we can't use either τ_1 or τ_2 (since relations $\mathcal{V}_{\tau_1}$ and $\mathcal{V}_{\tau_2}$ require both elements of the pair to be of type τ_1 or τ_2 respectively).

We address this issue by extending our notion of relational substitution, so that it maps a type variable to not just a pair of types, but also a relation on pairs of values of those types.

$$\begin{aligned}
\mathcal{V}^{\rho}_{\forall X.\ \tau} &= \{(\Lambda X.\, e_1, \Lambda X.\, e_2) \mid \forall \tau_1, \tau_2, R \in \mathrm{Rel}[\tau_1, \tau_2].\ (e_1\{\tau_1/X\}, e_2\{\tau_2/X\}) \in \mathcal{E}^{\rho[X \mapsto (\tau_1, \tau_2, R)]}_{\tau}\} \\
\mathcal{V}^{\rho}_{X} &= \{(v_1, v_2) \mid \rho(X) = (\tau_1, \tau_2, R) \text{ and } (v_1, v_2) \in R\}
\end{aligned}$$

With a few notional conventions, we will be ready to give our final and correct versions of the logical relations. For a relational substitution ρ we write ρ_1, ρ_2, and ρ_R for the first, second, and third projections. E.g., if $\rho(X) = (\tau, \tau', S)$, then $\rho_1(X) = \tau$, $\rho_2(X) = \tau'$, and $\rho_R(X) = S$. We write $\rho_1(\tau)$ for the result of replacing free type variables in τ using substitution ρ_1, and similarly for $\rho_2(\tau)$.

$$\begin{aligned}
\mathcal{V}^{\rho}_{\textbf{int}} &= \{(n, n) \mid n \in \mathbb{Z}\} \\
\mathcal{V}^{\rho}_{\tau \to \tau'} &= \{(\lambda x\!:\!\rho_1(\tau).\, e_1, \lambda x\!:\!\rho_2(\tau).\, e_2) \mid \forall (v_1, v_2) \in \mathcal{V}^{\rho}_{\tau}.\ (e_1\{v_1/x\}, e_2\{v_2/x\}) \in \mathcal{E}^{\rho}_{\tau'}\} \\
\mathcal{V}^{\rho}_{\forall X.\ \tau} &= \{(\Lambda X.\, e_1, \Lambda X.\, e_2) \mid \forall \tau_1, \tau_2, R \in \mathrm{Rel}[\tau_1, \tau_2].\ (e_1\{\tau_1/X\}, e_2\{\tau_2/X\}) \in \mathcal{E}^{\rho[X \mapsto (\tau_1, \tau_2, R)]}_{\tau}\} \\
\mathcal{V}^{\rho}_{X} &= \{(v_1, v_2) \mid \rho(X) = (\tau_1, \tau_2, R) \text{ and } (v_1, v_2) \in R\}
\end{aligned}$$

We define the relations for expressions by requiring that the expressions terminate in values in the appropriate value relation.

$$\begin{aligned}
\mathcal{E}^{\rho}_{\tau} = \{(e_1, e_2) \mid \quad & \vdash e_1 \!:\! \rho_1(\tau) \text{ and } \vdash e_2 \!:\! \rho_2(\tau) \text{ and} \\
& \exists v_1, v_2.\ e_1 \longrightarrow^* v_1 \text{ and } e_2 \longrightarrow^* v_2 \text{ and } (v_1, v_2) \in \mathcal{V}^{\rho}_{\tau}\}
\end{aligned}$$

We would now like to define our logical relation that will imply contextual equivalence. Since contextual equivalence is defined for expressions with free variables (see Definition 1), our logical relation will be as well. We thus need to provide an interpretation for type contexts Δ and variable contexts Γ. (We write $\bullet$ for an empty type context or variable context.)

$$\begin{aligned}
\mathcal{D}[\![\bullet]\!] &= \{\emptyset\} \\
\mathcal{D}[\![\Delta, X]\!] &= \{\rho[X \mapsto (\tau_1, \tau_2, R)] \mid \rho \in \mathcal{D}[\![\Delta]\!] \text{ and } R \in \mathrm{Rel}[\tau_1, \tau_2]\} \\
\mathcal{G}[\![\bullet]\!]_\rho &= \{\emptyset\} \\
\mathcal{G}[\![\Gamma, x\!:\!\tau]\!]_\rho &= \{\gamma[x \mapsto (v_1, v_2)] \mid \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho \text{ and } (v_1, v_2) \in \mathcal{V}^\rho_\tau\}
\end{aligned}$$

Note that the interpretation of variable context Γ requires a relational substitution ρ, because τ might have free type variables. For relational type variable substitutions γ, we also use γ_1 and γ_2 to project the first and second element of the pairs of types respectively, and write $\gamma_1(e)$ for the result of replacing free variables in e using γ_1, and similarly for γ_2.

We now define our logical relation.

$$\begin{aligned}
\Delta, \Gamma \vdash e_1 \approx e_2 : \tau \triangleq{} & \Delta, \Gamma \vdash e_1 : \tau \text{ and } \Delta, \Gamma \vdash e_2 : \tau \text{ and} \\
& \forall \rho \in \mathcal{D}[\![\Delta]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho.\ \ (\ \rho_1(\gamma_1(e_1))\ ,\ \rho_2(\gamma_2(e_2))\ ) \in \mathcal{E}^\rho_\tau
\end{aligned}$$

Intuitively, we require for any relations substitutions ρ and γ that are consistent with Δ and Γ, we have that e_1 and e_2 (with appropriate substitutions for free variables and type variables) are in the relation $\mathcal{E}^\rho_\tau$.