Canonicity

Alex Kavvos

Reading: arXiv:1907.11133

Recall the following special case of a property of the simply-typed λ-calculus:

Theorem 1 (Canonicity). For every $\vdash e : \mathsf{Num}$ there exists a v val such that $e \longmapsto^* v$.

Induction does not suffice to prove it. The reasons are somewhat deep. However, we are able to prove it through the technique of **logical relations**. Recall that a unary relation is called a **predicate**.

1 Outline

Consider the STLC without strings. Define a predicate $e \in P_\tau$ on pre-terms by **induction on types**.

$$\begin{aligned} e \in P_{\mathsf{Num}} &\equiv \exists v.\ \vdash v : \mathsf{Num} \wedge v\ \mathsf{val} \wedge e \longmapsto^* v \\ e \in P_{\sigma \times \tau} &\equiv \pi_1(e) \in P_\sigma \wedge \pi_2(e) \in P_\sigma \\ e_1 \in P_{\sigma \to \tau} &\equiv \forall e_2 \in P_\sigma.\ e_1(e_2) \in P_\tau \end{aligned}$$

We will prove the following result.

Lemma 2. If $\vdash e : \tau$ then $e \in P_\tau$.

Consequently, if $\Gamma \vdash e : \mathsf{Num}$ then $e \in P_{\mathsf{Num}}$. Thus there exists a numerical v val with $e \longmapsto^* v$.

2 Substitutions

Unfortunately, Lemma 2 is not strong enough to be proved by induction. We need to **strengthen the IH**.

Let $x, y, z, \ldots \in \mathcal{V}$ be the set of variables.

A **substitution** is a finite map $\gamma : \mathcal{V} \rightharpoonup \mathsf{PreTerm}$ mapping variables to pre-terms.

We define $e[\gamma]$ inductively as before; for example

$$\begin{aligned} x[\gamma] &\stackrel{\text{def}}{\simeq} \gamma(x) \\ (e_1(e_2))[\gamma] &\stackrel{\text{def}}{\simeq} e_1[\gamma](e_2[\gamma]) \\ &\vdots \end{aligned}$$

Finally, given a context Γ define

$$\gamma \vDash \Gamma \quad \stackrel{\text{def}}{\equiv} \quad \forall (x : \sigma) \in \Gamma.\ \gamma(x) \in P_\sigma$$

We will then prove

Lemma 3. If $\Gamma \vdash e : \tau$ and $\gamma \vDash \Gamma$ then $e[\gamma] \in P_\tau$.

From this Lemma 2 follows by picking Γ to be the empty context.

What is more, this can be shown by induction!

3 Some cases of the proof

First, another lemma:

Lemma 4. If $e_1 \longmapsto e_2$ and $e_2 \in P_\sigma$ then $e_1 \in P_\sigma$.

Proof. By induction on σ. $\square$

We can then produce a

Proof of Lemma 2. By induction on the derivation of $\Gamma \vdash e : \tau$.

Case(Var). Suppose the derivation is $\Gamma, x : \tau \vdash x : \tau$, so that $e = x$. Then from $\gamma \vDash \Gamma$ we know that $\gamma(x) \in P_\sigma$. But from the definition of substitution we have $e[\gamma] = x[\gamma] \stackrel{\text{def}}{=} \gamma(x)$, which is then in the relation.

Case(App). Suppose the derivation is of the form

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash e_1 : \sigma \to \tau} \qquad \dfrac{\vdots}{\Gamma \vdash e_2 : \sigma}}{\Gamma \vdash e_1(e_2) : \tau}\ \textsc{App}$$

By the IH, we have that $e_1[\gamma] \in P_{\sigma\to\tau}$ and $e_2[\gamma] \in P_\sigma$.

By the definition of $P_{\sigma\to\tau}$ we then have that $e_1[\gamma](e_2[\gamma]) \in P_\tau$. But $(e_1(e_2))[\gamma] \stackrel{\text{def}}{=} e_1[\gamma](e_2[\gamma])$, so we are done.

Case(Lam). Suppose the derivation is of the form

$$\dfrac{\dfrac{\vdots}{\Gamma, x : \sigma \vdash u : \tau}}{\Gamma \vdash \lambda x : \sigma.\, u : \sigma \to \tau}\ \textsc{Lam}$$

We need to show that $(\lambda x : \sigma.\, u)[\gamma] \stackrel{\text{def}}{=} \lambda x : \sigma.\, u[\gamma] \in P_{\sigma\to\tau}$.

By definition, this means that assuming $e \in P_\sigma$ we have to show $(\lambda x : \sigma.\, u[\gamma])(e) \in P_\tau$.

So assume $e \in P_\sigma$. By D-Beta we have

$$(\lambda x : \sigma.\, u[\gamma])(e) \longmapsto u[\gamma][e/x] \equiv u[\gamma'] \tag{$*$}$$

where

$$\gamma'(z) \simeq \begin{cases} e & \text{if } z = x \\ \gamma(z) & \text{otherwise} \end{cases}$$

Notice that $\gamma' \vDash \Gamma, x : \sigma$, as x is mapped to $e \in P_\sigma$. Hence by the IH we have $u[\gamma'] \in P_\tau$.

Therefore by Lemma 4 and $(*)$ we have $(\lambda x : \sigma.\, u[\gamma])(e) \in P_\tau$.

$\square$

Note that the cases of operations on ground types (e.g. $\mathsf{plus}(-;-)$) are somewhat annoying, as they depend on various admissible rules for $e_1 \longmapsto^* e_2$ which need to be proved by induction.

The method of logical relations is extremely general. It can be adapted to prove a host of properties, including type safety, noninterference, equivalence of programs, and so on. Moreover, it is extensible to languages with a higher-order store, polymorphism, and so on.