

Secure Compilation

Lecture 2

Closure Conversion

Renate Robin Eilers Cristina Matache Baber Rehman

June 24, 2019

This is the second talk presented by Amal Ahmed in OPLSS 2019, University of Oregon, USA.

1 Source Language

Types We just have integers and functions in source language.

$$\sigma ::= \text{int} \mid \sigma_1 \rightarrow \sigma_2$$

Terms

$$\begin{aligned} v &::= x \mid n \mid \lambda x : \sigma. e \\ e &::= v \mid \text{if0 } v \ e_1 \ e_2 \mid v_1 \ v_2 \mid \text{let } x = e_1 \text{ in } e_2 \end{aligned}$$

So $e_1 \ e_2$ is a shorthand for $\text{let } x = e_1 \text{ in let } y = e_2 \text{ in } x \ y$.

Evaluation contexts:

$$E ::= [\cdot] \mid \text{let } x = E \text{ in } e_2$$

The language has a typing judgement $\Gamma \vdash e : \sigma$ and a small-step call-by-value operational semantics $e \mapsto e'$.

2 Target Language

Types and terms

$$\begin{aligned} \tau &::= \text{int} \mid (\tau_1, \dots, \tau_n) \rightarrow \tau' \mid \langle \tau_1, \dots, \tau_n \rangle \mid \alpha \mid \exists \alpha. \tau \\ v &::= x \mid n \mid \lambda(\overline{x} : \overline{\tau}). e \mid \langle v_1, \dots, v_n \rangle \mid \text{pack}(\tau, v) \text{ as } \exists \alpha. \tau \\ e &::= v \mid \text{if0 } v \ e_1 \ e_2 \mid v_1 \ (\overrightarrow{v}) \mid \pi_i \ v \mid \text{unpack}(\alpha, x) = v \text{ in } e_1 \mid \text{let } x = e_1 \text{ in } e_2 \end{aligned}$$

Typing contexts:

$$\begin{aligned}\Delta &::= \cdot \mid \Delta, \alpha \\ \Gamma &::= \cdot \mid \Gamma, x : \tau\end{aligned}$$

Typing judgements: $\Delta, \Gamma \vdash e : \tau$

To do closure conversion, we want functions to have a closed body:

$$\frac{\frac{\cdot \mid \overline{x : \tau} \vdash e : \tau'}{\Delta; \Gamma \vdash \lambda(\overline{x : \tau}).e : (\overrightarrow{\tau}) \rightarrow \tau'}}{\Delta; \Gamma \vdash v : \tau[\tau'/\alpha]} \quad \frac{}{\Delta; \Gamma \vdash \text{pack}(\tau', v) \text{ as } \exists \alpha. \tau : \exists \alpha. \tau}$$

Example 1. A term of type $\exists \alpha. \alpha \times (\alpha \rightarrow \text{int})$ is:

$$w = \text{pack}(\text{bool}, \langle \text{true}, \lambda x : \text{bool}. 5 \rangle) \text{ as } \exists \alpha. \alpha \times (\alpha \rightarrow \text{int})$$

$$\frac{\Delta; \Gamma \vdash v : \exists \alpha. \tau \quad \Delta, \alpha; \Gamma, x : \tau \vdash e_2 : \tau_2 \quad \Delta \vdash \tau_2}{\Delta; \Gamma \vdash \text{unpack}(\alpha, x) = v \text{ in } e_2 : \tau_2}$$

In the rule above α is not allowed to appear in τ_2 .

Example 2. A well-typed term is:

$$\text{unpack}(\alpha, x) = w \text{ in } (\pi_2 x) (\pi_1 x)$$

where w is defined as in the previous example.

3 Translation

Translation of types: σ^+

$$\begin{aligned}\text{int}_S^+ &= \text{int}_T \\ (\sigma_1 \rightarrow \sigma_2)^+ &= \exists \alpha_{\text{env}}. \langle (\alpha_{\text{env}}, \sigma_1^+) \rightarrow \sigma_2^+, \alpha_{\text{env}} \rangle\end{aligned}$$

Typing context translation: Γ_S^+

$$\begin{aligned}(\cdot)^+ &= \cdot \\ (\Gamma_S, x_S : \sigma)^+ &= \Gamma_S^+, x_T : \sigma^+\end{aligned}$$

Term translation: $\Gamma_S \vdash e_S : \sigma \rightsquigarrow e_T$ where $;\Gamma_S^+ \vdash e_T : \sigma^+$

$$\frac{\Gamma_S(x_S) = \sigma}{\Gamma_S \vdash x_S : \sigma \rightsquigarrow x_T} \quad \frac{}{\Gamma_S \vdash n_S : \text{ints}_S \rightsquigarrow n_T}$$

$$\frac{y_{S_1}, \dots, y_{S_n} = \text{free variables}(\lambda x_S : \sigma.e_S) \quad \Gamma_S \vdash y_{S_i} : \sigma_i \quad v_{code} = \lambda(z_T : \langle \sigma_1^+, \dots, \sigma_n^+ \rangle).e_T[(\pi_i z)/y_{T_i}]}{\Gamma_S \vdash \lambda x_S : \sigma.e_S : \sigma \rightarrow \sigma' \rightsquigarrow \text{pack}(\langle \sigma_1^+, \dots, \sigma_n^+ \rangle, \langle v_{code}, \langle y_{T_1}, \dots, y_{T_n} \rangle \rangle) \text{ as } (\sigma \rightarrow \sigma')^+}$$

where $e_T[(\pi_i z)/y_{T_i}]$ is a shorthand for

$$\text{let } y_{T_1} = \pi_1 z \text{ in } \dots \text{let } y_{T_n} = \pi_n z \text{ in } e_T$$

$$\frac{\Gamma_S \vdash v_{S_2} : \sigma_2 \rightsquigarrow v_{T_2} \quad \Gamma_S \vdash v_{S_1} : \sigma_2 \rightarrow \sigma \rightsquigarrow v_{T_1}}{\Gamma_S \vdash v_{S_1} v_{S_2} : \sigma \rightsquigarrow \text{unpack}(\alpha, p) = v_{T_1} \text{ in } (\pi_1 p) (\pi_2 p, v_{T_2})}$$

The rules for `if0` and `let` are defined according to the structure of the terms.

4 Preservation Theorem

Theorem 4.1 (Type Preservation). *If $\Gamma \vdash e_S : \sigma$ and $\Gamma \vdash e_S : \alpha \rightsquigarrow e_T$ then $\Gamma_S^+ \vdash e_T : \sigma^+$*

For correctness, we want to show $e_S \approx e_T$. This is not contextual equivalence because source language and target language are two different languages. There are many ways to prove compiler correction. We want to say that when:

$$e_S \approx e_T \text{ then } \sigma \approx \sigma^+$$

$$\begin{aligned} V[\![\sigma]\!] &= \{ (V_S, V_T) \mid j \cdot \vdash V_S : \sigma \wedge \cdot ; \cdot \vdash V_T : \sigma^+ \dots \} \\ V[\![\text{ints}]\!] &= \{ (n_S, n_T) \} \\ V[\![\sigma_1 \rightarrow \sigma_2]\!] &= \{ (\lambda x : \sigma_1 \cdot e_S \text{ pack } (\tau_{env}, \langle \lambda (Z : \tau, x_T : \sigma_1^+) \cdot e_T, V_{env} \rangle)) \\ j \forall (v_S, v_T) \mathcal{E} \ V[\![\sigma_1]\!] \cdot (e_S[v_S / x_S], e_T[v_{env} / z, v_T / x_T]) \in \mathcal{E}[\![\sigma_2]\!] \} \end{aligned}$$

5 Logical Relations

In logical relations we map related input to related outputs. Same source value and target value are related.