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 function in source language.

$$\sigma ::= int \mid \sigma_1 \to \sigma_2$$

Terms

$$v ::= x \mid n \mid \lambda x : \sigma . e$$
$$e ::= v \mid \texttt{if0}\ v\ e_1\ e_2 \mid v_1\ v_2 \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2$$

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

Evaluation contexts:

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

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

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

Typing contexts:

$$\Delta ::= \cdot \mid \Delta, \alpha$$
$$\Gamma ::= \cdot \mid \Gamma, x : \tau$$

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

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

$$\frac{\cdot \mid \overline{x : \tau} \vdash e : \tau'}{\Delta; \Gamma \vdash \lambda(\overline{x : \tau}).e : (\overrightarrow{\tau}) \to \tau'}$$

$$\frac{\Delta; \Gamma \vdash v : \tau[\tau'/\alpha]}{\Delta; \Gamma \vdash \texttt{pack}(\tau',\ v)\ \texttt{as}\ \exists\alpha.\tau : \exists\alpha.\tau}$$

Example 1. *A term of type* $\exists\alpha.\ \alpha \times (\alpha \to int)$ *is:*

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

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

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

Example 2. *A well-typed term is:*

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

where w *is defined as in the previous example.*

3 Translation

Translation of types: σ^+

$$int_S^+ = int_T$$
$$(\sigma_1 \to \sigma_2)^+ = \exists\alpha_{env}.\ \langle(\alpha_{env},\ \sigma_1^+) \to \sigma_2^+,\ \alpha_{env}\rangle$$

Typing context translation: Γ_S^+

$$(\cdot)^+ = \cdot$$
$$(\Gamma_S, x_S : \sigma)^+ = \Gamma_S^+, x_T : \sigma^+$$

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

$$\frac{\Gamma_S(x_S) = \sigma}{\Gamma_S \vdash x_S : \sigma \rightsquigarrow x_T} \qquad \frac{}{\Gamma_S \vdash n_S : int_S \rightsquigarrow n_T}$$

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

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

$$\texttt{let}\ y_{T_1} = \pi_1\ z\ \texttt{in}\ \ldots \texttt{let}\ y_{T_n} = \pi_n\ z\ \texttt{in}\ e_T$$

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

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 contexual equivalance because source language and target language are two different languages. There are many ways to prove compiler correction. We want to say that:

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

5 Logical Relations

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

Values $\mathrm{V}[\![\sigma]\!] = \{(v_S\ ,\ v_T) \mid \cdot \vdash v_S{:}\sigma \wedge \cdot\ ; \cdot \vdash v_T{:}\sigma^+\ .....\ \}$

Integers $\mathrm{V}[\![\text{ints}]\!] = \{(n_S\ ,\ n_T)\}$

Function $\mathrm{V}[\![\sigma_1{\to}\sigma_2]\!] = \{(\lambda\text{x}{:}\sigma_1 \cdot e_S\ \texttt{pack}\ (\tau_{env}\ ,\ \langle\ \lambda\ (\text{Z}{:}\tau{,}x_T{:}\sigma_1^+) \cdot e_T{,}V_{env}\ \rangle\ )\ \mid \forall(v_S\ ,\ v_T)\ \mathcal{E}\ \mathrm{V}[\![\sigma_1]\!]\ .\ (e_S[v_s/x_s]{,}e_T\ [v_{env}/\text{z}{,}v_T/x_T]) \in \mathcal{E}\ [\![\sigma_2]\!]\}$

Typing Judgement $\mathcal{E}\ [\![\sigma]\!] = \{(e_S,e_T) \cdot \vdash e_S{:}\sigma \wedge \cdot ; \vdash e_T{:}\sigma^+ \wedge \forall v_S \cdot e_S \longmapsto^* v_S \Rightarrow \exists v_T \cdot e_T \rightarrow^* v_T \wedge (v_S\ , v_T) \in \text{v}\ [\![\sigma]\!] \wedge \forall v_T \cdot e_T \longmapsto^* v_T \Rightarrow \exists v_S \cdot e_S \longmapsto^* v_S \wedge (v_S,v_T) \in \text{v}\ [\![\sigma]\!]\}$

Target language behavious is shown in source language.

Definition 5.0.1. $\Gamma_S \vdash e_S \approx e_T{:}\sigma\ = \Gamma_S \vdash e_S{:}\sigma \wedge \cdot\ ;\ \Gamma_S^+ \vdash e_T{:}\sigma^+ \wedge \forall(\gamma_S,\gamma_T) \in \mathcal{G}\ [\![\sigma]\!] \cdot (\gamma_S(e_S),\gamma_T(e_T)) \in \mathcal{E}\ [\![\sigma]\!]$.

S and T are like holes in expression. They are not complete program. This is like substitution and linking. It will be combined with other code.

$\gamma_S\ = \{x_{S1} \longmapsto v_{S1}\ ....\ \}$
$\gamma_T\ = \{x_{T1} \longmapsto v_{T1}\ ....\ \}$

$\mathcal{G}[\![\cdot]\!]\ = \{\phi \cdot \phi\}$
$\mathcal{G}[\![x_S{:}\sigma]\!]\ = \{(\gamma_S[x_S \longmapsto v_S]\ ,\ \gamma_T[x_T \longmapsto v_T])\ |\ (\gamma_S,\gamma_T) \in \mathcal{G}[\![x_S{:}\Gamma]\!] \wedge\ (v_S,v_T) \in\ V[\![x_S{:}\sigma]\!]\}$

Theorem 5.1 (Compiler Correctness)**.** *If* $\Gamma_S \vdash e_S{:}\sigma \rightsquigarrow e_T$ *then* $\Gamma \vdash e_S \approx e_T{:}\sigma$.

Proof. This theorem can be proved by induction on typing derivation of source language. It can be proved just by unfoalding the definitions. □

Lemma 5.2 (Fundamental Property)**.** $\Gamma \vdash e_S{:}\sigma \Rightarrow \Gamma \vdash e_S \approx e_T{:}\sigma$