Lecture Notes on Resource Analysis

Part 3

JIAWEN LIU, JÉRÉMY THIBAULT, and SHAOWEI ZHU

These are lecture notes taken during Jan Hoffmann's course on resource analysis at the Oregon Programming Language Summer School 2019.

1 SHORT RECAP

We previously added the "tick" construct to the language:

$$\textit{<Expressions>}\ e ::= \cdots \mid \texttt{tick}\{q\}\,(e) \qquad \texttt{tick}\ q\ \texttt{in}\ e$$

Tick allows us to specify the amount of resource consumed at a finer grain.

We have also defined small-step semantics for the language $< e, q > \mapsto < e', q' >$ and a typing judgment, $\Gamma \vdash^{q}_{q'} e : \tau$.

More Details on the Typing Rule for Lambda Abstractions. Remember the rule for typing lambda abstractions:

$$\frac{\Gamma, x : \tau \vdash^{p}_{p'} e : \tau' \qquad |\Gamma| = \Gamma}{\Gamma \vdash^{0}_{0} \texttt{lam}\{\tau\}\,(x.e) : \tau \xrightarrow{p|p'} \tau'}$$

Here we formally define the operation $|\cdot|$ (remember that this roughly says that we cannot draw resource from the environment).

Definition 1.1 ($|\cdot|$). The functions $|\cdot|_\tau$ and $|\cdot|_A$ are defined mutually as follows:

$$|\texttt{unit}|_\tau = \texttt{unit}$$
$$|A \to B|_\tau = A \to B$$
$$|L(A)|_\tau = L(|A|_A)$$
$$|< \tau, q >|_A = < |\tau|_\tau, 0 >$$

The operation is extended to environments, in a point-wise manner:

$$|\cdot| = \cdot$$
$$|x : \tau, \Gamma| = x : |\tau|_\tau, |\Gamma|$$

Example 1.1. Consider the function f defined by

$$f = \lambda\,(x : L(\texttt{unit}))\,.\,\lambda\,(y : L(\texttt{unit}))\,.\,x$$
$$f : L^1(\texttt{unit}) \xrightarrow{0|0} L^0(\texttt{unit}) \xrightarrow{0|0} L^1(\texttt{unit})$$

This function f cannot be typed this way: it is prohibited by the rule for typing lambda-abstractions.

Authors' address: Jiawen Liu, jliu223@buffalo.edu; Jérémy Thibault, jeremy.thibault@inria.fr; Shaowei Zhu, shaoweiz@cs.princeton.edu.

$$\frac{\Gamma \vdash^{q}_{q'} e : \tau}{\Gamma \vdash^{q+p}_{q'} \texttt{tick}\{p\}\,(e) : \tau} \qquad \frac{}{\Gamma \vdash^{0}_{0} \mathrm{nil}\{\tau\} : L^{q}(\tau)} \qquad \frac{\Gamma_1 \vdash^{q}_{r} e_1 : \tau \qquad \Gamma_2 \vdash^{r}_{q'} e_2 : L^{p}(\tau)}{\Gamma_1, \Gamma_2 \vdash^{q+p}_{q'} \mathrm{cons}(e_1, e_2) : L^{p}(\tau)}$$

$$\frac{\Gamma_1 \vdash^{q}_{r} e : L^{p}(\tau) \qquad \Gamma_2 \vdash^{r}_{q'} e_1 : \tau' \qquad \Gamma_2, x_1 : \tau, x_2 : L^{p}(\tau) \vdash^{r+p}_{q'} e_2 : \tau'}{\Gamma_1, \Gamma_2 \vdash^{q}_{q'} \texttt{matL}\,(e, e_1, x_1.x_2.e_2) : \tau'}$$

Fig. 1. Typing rules for lists and tick

This is justified, as allowing such type would lead to an incorrect result. Consider

$$g = f(\texttt{triv} :: \texttt{triv} :: \mathrm{nil})$$

and suppose there is a function $h : L^1(\texttt{unit}) \xrightarrow{0|0} L^0(\texttt{unit})$.

Then, one would obtain, for the following computation:

$$< h(g(\mathrm{nil})), h(g(\mathrm{nil})) >$$

a bound of 2 instead of the expected 4.

The last typing rules for lists and tick can be found in Fig. 1. Note that in the rule for `cons`, below the line we need $q + p$ here to make sure that the head is not getting potential from nowhere. Also in the rule for `matL`, above the line we are allowed to use $r + p$ potential to type e_2 since the head of the list carries p potential.

2 STRUCTURAL RULES AND SUBTYPING

In this section, we introduce structural rules and subtyping. These kinds of rules allow us to type more programs without losing soundness.

2.1 Structural Rules

Example 2.1 (Motivating Example). Here is a simple type derivation:

$$\cfrac{\cfrac{x : L^{10}(\texttt{unit}) \vdash^{0}_{0} x : L^{10}(\texttt{unit})}{x : L^{10}(\texttt{unit}) \vdash^{5}_{0} \texttt{tick}\{5\}\,(x) : L^{10}(\texttt{unit})}}{\cdot \vdash^{0}_{0} \lambda\,(x : L(\texttt{unit}))\,.\,\texttt{tick}\{5\}\,(x) : L^{10}(\texttt{unit}) \xrightarrow{5|0} L^{10}(\texttt{unit})}$$

In this type derivation, the first level must be obtained by weakening. This is necessary, since we cannot apply directly the nil rule.

Similarly, for the following type derivation, we also need weakening:

$$\cfrac{\cfrac{\cfrac{\cdot \vdash^{0}_{0} [] : L^{10}(\texttt{unit})}{x : L^{10}(\texttt{unit}) \vdash^{0}_{0} [] : L^{10}(\texttt{unit})}}{x : L^{10}(\texttt{unit}) \vdash^{5}_{0} \texttt{tick}\{5\}\,([]) : L^{10}(\texttt{unit})}}{\cdot \vdash^{0}_{0} \lambda\,(x : L(\texttt{unit}))\,.\,\texttt{tick}\{5\}\,([]) : L^{10}(\texttt{unit}) \xrightarrow{5|0} L^{10}(\texttt{unit})}$$

To type these programs properly, we introduce the following two rules, the weakening rule and the relaxing rule:

$$\textsc{Weak}\ \frac{\Gamma \vdash^{q}_{q'} e : \tau'}{\Gamma, x : \tau \vdash^{q}_{q'} e : \tau'} \qquad \textsc{Relax}\ \frac{\Gamma \vdash^{p}_{p'} e : \tau' \qquad q \geq p \qquad q - q' \geq p - p'}{\Gamma \vdash^{q}_{q'} e : \tau'}$$

The weakening rule states that one might add irrelevant things into the context, without effects on the result.

The relaxing rule states that it is possible to start with and consume a little more resources than required.

2.2 Subtyping

Subtyping can be added to the language. The rules are as follow:

$$\frac{\tau <: \tau' \qquad \Gamma \vdash^{q}_{q'} e : \tau}{\Gamma \vdash^{q}_{q'} e : \tau'} \qquad \frac{\tau <: \tau'' \qquad \Gamma, x : \tau'' \vdash^{q}_{q'} e : \tau'}{\Gamma, x : \tau \vdash^{q}_{q'} e : \tau'} \qquad \frac{\tau <: \tau' \qquad q \geq q'}{< \tau, q > <: < \tau', q' >} \qquad \frac{}{\texttt{unit} <: \texttt{unit}}$$

$$\frac{A <: B}{L(A) <: L(B)} \qquad \frac{A_2 <: A_1 \qquad B_1 <: B_2}{A_1 \to B_1 <: A_2 \to B_2}$$

These subtyping rules are the standard rules for subtyping. In particular, note that a type with a potential is a subtype of a type with a potential that is smaller: everything that accept some argument with some potential can also accept one that has less.

3 SHARING

The current type system still has limitations, as shown by the following example.

Example 3.1 (Motivating example for sharing). Let us try to define and type the following function:

$$\begin{aligned} f = \text{fix double} : L(\texttt{unit}) &\to L(\texttt{unit}) \text{ as} \\ &\lambda x.\ \text{match } x \text{ with} \\ &\quad [] \mapsto [] \\ | &\quad y :: ys \mapsto y :: y :: \text{double}(ys) \end{aligned}$$

Here, the affine type system is having trouble typing the function, as y is used more than once.

To solve this issue, we introduce a new construct, "sharing". For instance, we can now write:

$$\lambda\,(x : L(\texttt{unit}))\,.\ \texttt{share}\ x\ \texttt{as}\ x_1, x_2\ \texttt{in}\ \text{id}(x_1) :: \text{id}(x_2) :: []$$

with the type

$$L^4(\texttt{unit}) \xrightarrow{0|0} L^0(\texttt{unit}).$$

We update our grammar accordingly:

$$e ::= \cdots \mid \texttt{share}\ e_1\ \texttt{as}\ x_1, x_2\ \texttt{in}\ e_2$$

$$\frac{\Gamma_1 \vdash^q_p e_1 : \tau \qquad \Gamma_2, x_1 : \tau_1, x_2 : \tau_2 \vdash^p_{q'} e_2 : \tau' \qquad \tau \curlyvee \tau_1 \mid \tau_2}{\Gamma_1, \Gamma_2 \vdash^q_{q'} \texttt{share}\ e_1\ \texttt{as}\ x_1, x_2\ \texttt{in}\ e_2 : \tau} \qquad \frac{}{A \to B \curlyvee A \to B \mid A \to B} \qquad \frac{A \curlyvee A_1 \mid A_2}{L(A) \curlyvee L(A_1) \mid L(A_2)}$$

$$\frac{\tau \curlyvee \tau_1 \mid \tau_2 \qquad q = q_1 + q_2}{< \tau, q > \curlyvee < \tau_1, q_1 >\mid \tau_2, q_2} \qquad \frac{}{\texttt{unit} \curlyvee \texttt{unit} \mid \texttt{unit}}$$

Fig. 2. Updated rules for sharing

and the semantics:

$$\frac{e_1 \mapsto e_1'}{\texttt{share}\ e_1\ \texttt{as}\ x_1, x_2\ \texttt{in}\ e_2 \mapsto \texttt{share}\ e_1'\ \texttt{as}\ x_1, x_2\ \texttt{in}\ e_2} \qquad \frac{e_1\ \text{val}}{\texttt{share}\ e_1\ \texttt{as}\ x_1, x_2\ \texttt{in}\ e_2 \mapsto [e_1, e_1/x_1, x_2]e_2}$$

The static semantics have to be update by introducing a new symbol, $\curlyvee$. The judgment $\tau \curlyvee \tau_1 \mid \tau_2$ reads "τ is shared as τ_1 and τ_2". We give the new typing rule and the inductive definition of sharing in Fig. 2

The rules that define sharing enforce that no resource is created or lost when sharing.

Example 3.2.

$$L^4(L^2(\texttt{unit})) \curlyvee L^2(L^2(\texttt{unit})) \mid L^2(L^0(\texttt{unit}))$$

Homework. Give a type derivation for

$$f = \lambda\,(x : L(\texttt{unit}))\,.\,\text{id}(\text{double}(x))$$

Finally, we can still state *progress* and *preservation* theorems for this type system.

Theorem 3.3 (Progress). *If* $\vdash^q_{q'} e : \tau$ *and* $p \geq q$ *then either* e *val or* $< e, p > \mapsto < e', p' >$ *for some* $< e', p' >$.

Theorem 3.4 (Preservation). *If* $\vdash^q_{q'} e : \tau, p \geq q$, *and* $< e, p > \mapsto < e', p' >$ *then* $\vdash^{p'}_{q'} e' : \tau$.

While the proof of progress is simple, the proof of preservation is complicated (involves a nested induction on the typing judgment and the semantic stepping).