Resource Analysis: Lecture 3

JAN HOFFMANN

1 SHORT RECAPITULATION

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

⟨*Expression*⟩ e ::= …

| tick{q} (e)

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:

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

Here we formally define the operation $|\cdot|$.

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

$$\begin{aligned} |\text{unit}|_\tau &= \text{unit} \\ |A \to B|_\tau &= A \to B \\ |L(A)|_\tau &= L(|A|_A) \\ |<\tau, q>|_A &= <|\tau|_\tau, 0> \end{aligned}$$

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

$$\begin{aligned} |\cdot| &= \cdot \\ |x : \tau, \Gamma| &= x : |\tau|_\tau, |\Gamma| \end{aligned}$$

Example 1.2. Consider the function f defined by

$$\begin{aligned} f &= \lambda\,(x : L(\text{unit}))\,.\,\lambda\,(y : L(\text{unit}))\,.\,x \\ f &: L^1(\text{unit}) \xrightarrow{0|0} L^0(\text{unit}) \xrightarrow{0|0} L^1(\text{unit}) \end{aligned}$$

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

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

$$g = f(\langle\rangle :: \langle\rangle :: \text{nil})$$

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

Then, one would obtain, for the following computation:

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

Author's address: Jan Hoffmann, jhoffmann@cmu.edu.

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

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

Fig. 1. Typing rules for lists and tick

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

Fig. 2. Structural Rules

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

2.1 Structural Rules

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

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

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:

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

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.

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

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

Fig. 3. Rules for subtyping

2.2 Subtyping

Subtyping can be added to the language. The rules are given in Fig. 3.

3 SHARING

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

Example 3.1. Let us try to define and type the following function:

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

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:

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

with the type

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

We update our grammar accordingly:

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

and the semantics:

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

The static semantics have to be update by introducing a new symbol, †. The judgment $\tau \dagger \tau_1 \mid \tau_2$ reads "τ is shared as τ_1 and τ_2". We give the new typing rule and the inductive definition of sharing in Fig. 4

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

Example 3.2.

$$L^4(L^2(\text{unit})) \dagger L^2(L^2(\text{unit})) \mid L^2(L^0(\text{unit}))$$

$$\frac{\Gamma_1 \vdash^{q}_{p} e_1 : \tau \qquad \Gamma_2, x_1 : \tau_1, x_2 : \tau_2 \vdash^{p}_{q'} e_2 : \tau' \qquad \tau \dagger \tau_1 \mid \tau_2}{\Gamma_1, \Gamma_2 \vdash^{q}_{q'} \text{share } e_1 \text{ as } x_1, x_2 \text{ in } e_2 : \tau} \qquad \frac{}{A \to B \dagger A \to B \mid A \to B} \qquad \frac{A \dagger A_1 \mid A_2}{L(A) \dagger L(A_1) \mid L(A_2)}$$

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

Fig. 4. Updated rules for sharing

Homework. Give a type derivation for

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

Finally, we can still state *progress* and *preservation* 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).