Resource Analysis
Lecture 4

Renate Robin Eilers    Cristina Matache    Baber Rehman

June 22, 2019

This is the fourth talk presented by Jan Hoffmann in OPLSS 2019, University of Oregon, USA.

1 Recap: Soundness

This talk is mainly about soundness proof, type inference, and examples in RAML. Talk started by a recap of progress and preservation which were formulated in last talk.

**Theorem 1.1** (Progress). If $\vdash q \quad e : \tau$ and $p \geq q$ then either $e$ is a value or $\exists e', p'$ s.t. $(e, p) \rightarrow (e', p')$.

**Theorem 1.2** (Preservation). If $\vdash q \quad e : \tau$, $p \geq q$ and $(e, p) \rightarrow (e', p')$ then $\vdash p' \quad e' : \tau$.

*Proof notes.* Preservation is difficult to prove. It is proved by nested induction on $\vdash q \quad e : \tau$ and $(e, p) \rightarrow (e', p')$. There are some tricky lemmas to prove like substitution which only holds on values.

*Alternative soundness theorem:* Recall the judgement $V \vdash e \Downarrow v \mid (q, q')$

**Definition 1.2.1.**

$$\phi(V : \Gamma) = \sum_{x \in \text{dom}(\Gamma)} \phi(V(x) : \Gamma(x))$$

where $\Gamma$ assigns types to variables.

**Theorem 1.3.** Let $V : \Gamma$ and $\vdash q \quad e : \tau$ and $V \vdash e \Downarrow v \mid (p, p')$ then $\phi(V : \Gamma) + q \geq p$ and $\phi(V : \Gamma) + q - \phi(v : \tau) - q' \geq p - p'$

This theorem shows that the type derivation is a certificate for bound correctness.

2 Type inference

**Example 1.** We want to find a derivation for:

$$\vdash_0 \text{fix}(\text{id} \cdot \lambda(x : L(\text{unit})): \text{matL}(x; \text{nil}; y, y \cdot \text{cons}(y; \text{tick}\{2\}(\text{id}(y)))) : L^2(\text{unit}) \rightarrow^{0/0} L^0(\text{unit})$$
See figure 1 for the derivation tree, where $e_{id} = fix(id, \lambda x : L(unit)) matL(x; nil; y, ys.cons(y; tick\{2\}(id)))$ and $\tau_{id} = L^p(unit) \rightarrow q/q' L^p(unit)$.

For type inference we need algorithmic (or syntax-directed) rules. We change all the typing rules to incorporate the structural rules:

**Example 2.**

$$
\frac{\Gamma, x : \tau \vdash q \quad q' \quad \tau \lessdot q'}{
\Gamma, x : \tau \vdash q'/q' \quad x : \tau'
}$$

Algorithm for type inference:

1. Infer usual types (without annotations), which results in a type derivation (like example in figure 1 with all annotations removed);
2. Add potential variables where a potential annotation is required, both in $\tau_{id}$ and in the derivation. See figure 2. Note the this step is only partially done in the figure;

$$\tau_{id} \text{ becomes } L^p(unit) \rightarrow q/q' L^p'(unit)$$

3. Derive from the typing rules linear constraints on potential variables;

**Example 3.** For the fix example, some of the constraints are:

- $r_0 \geq r'_0$
- $r_2 \leq q$
- $p_1 \leq p$
- $r_3 \geq r'_3$
- $p_2 \leq p_1$
- $r_1 = 0$
- $r'_2 \geq q'$
- $p'_1 \geq p'$
- $r_3 \leq r_2$
- $p'_2 \geq p_2$
- $r'_1 = 0$
- $r'_4 \leq r'_3 + p_1$
- $r'_4 < r'_2$
- $s_1 \geq s_2 + 2$
- $s'_1 = s'_2$

4. Solve constraints with LP solver;
5. Objective is the sum of initial potential annotations.

**Example 4.** For the fix example the objective is to minimize $p + q$.

### 3 Implementation in RAML and examples

Live RAML (Resource aware ML) demo showing binary counter, using the source code displayed in figure 3. Second example with queue.
Figure 1: Example of type inference
\[
\begin{align*}
\text{relax} & \quad \frac{x : L^{p_2}(1) \vdash y : 1}{y : 1 \vdash y : 1} \\
\text{app} & \quad \frac{id : \tau \vdash y : L(1) \vdash y : L(1)}{id : \tau \vdash \text{id}(ys) : L(1)} \\
& \quad \frac{id : \tau \vdash y : L(1) \vdash \text{tick}(\{2\}(id(ys))) : L(1)}{id : \tau \vdash \text{cons}(y, \text{tick}(\{2\}(id(ys)))) : L(1)}
\end{align*}
\]

Figure 2: Type inference, step 2 of the algorithm
let rec id x = 
  match x with
  | [] -> []
  | y::ys -> y::(let _ = Raml.tick 2.0 in ys)

type bit = Zero | One

let rec inc counter = 
  match counter with
  | [] -> [One]
  | Zero::bs -> One::bs
  | One::bs -> Zero::(inc bs)

let rec in_many n = 
  match n with
  | Z -> []
  | S n' -> inc (in_many n')

Figure 3: Code for binary counter example