Resource Analysis: Problem Set 7

Jan Hoffmann
Carnegie Mellon University

March 16, 2016

Due before 1:30pm on Monday, March 28

7.1 (15 Points) Functional Queue Revisited

In this problem we are interested in the number of cons operations. We use a metric M with $M^{\mathsf{cons}} = 1$ and $M^K = 0$ for all $K \neq$ cons.

a) Consider the function *rev_append* below.

```
let rec rev_append (l1,l2) =
  match l1 with
  | [] → l2
  | x::xs → rev_append (xs, x::l2)
```

Give a type derivation using the type system for linear amortized analysis that is defined in Figure 1.

b) Give a concise description of the set of all annotated function types that are derivable for the function *rev_append*.

c) Now consider the following implementation of the functional queue from problem set.

```
let rev l = rev_append (l,[])

let enqueue (queue, x) =
  match queue with
  | (inq,outq) →
     (x::inq, outq)

let rec dequeue (inq,outq) =
  match outq with
  | [] →
     begin
       match inq with
       | [] → (([],[]),-1)
       | x::xs → dequeue ([],rev inq)
     end
  | y::ys → ((inq, ys), y)
```

Give a derivable resource-annotated types for the functions *rev*, *enqueue*, and *dequeue*. (You don't have to provide the type derivation.)

d) Give resource-annotated type derivations for the functions *a* and *b* below.

```
let a =
  let qu = ([],[]) in
  let qu = enqueue (qu, 1) in
  let qu = enqueue (qu, 2) in
  let qu = enqueue (qu, 3) in
  dequeue qu

let b =
  let qu = ([],[]) in
  let qu = enqueue (qu, 1) in
  let qu = enqueue (qu, 2) in
  let qu = enqueue (qu, 3) in
  let _ = dequeue qu in
  dequeue qu
```

7.2 (33 Points) Typeable Functions

Consider the following first-order functions. As earlier, we use *tick* functions to define the resource usage. That means that we use a resource metric that defines cost q for operation *Raml.tick* q and cost 0 for all other operations.

```
let rec insert (x,l) =
  let _ = Raml.tick 1.0 in
  match l with
  | [] → [x]
  | y::ys →
     if (y:int) <= x then y::insert (x,ys)
     else x::y::ys

let rec isort l =
  let _ = Raml.tick 1.0 in
  match l with
  | [] → []
  | x::xs → insert (x, isort xs)

let rec omega x = omega x

let rec repeat (x,l) =
  let _ = Raml.tick 1.0 in
  match l with
  | [] → []
  | y::ys → x::(repeat (x,ys))

let rec append (l1,l2) =
  let _ = Raml.tick 1.0 in
  match l1 with
  | [] →  let _ = Raml.tick (-1.0) in l2
  | x::xs → let res = x::append (xs,l2) in
            let _ = Raml.tick (-1.0) in res
```

```
let rec append_all l =
  let _ = Raml.tick 1.0 in
  match l with
  | [] → []
  | x::xs → append (x, append_all xs)

let f1 l = repeat (l, append_all l)

let f2 l = if true then l else isort l

let f3 l = let l' = omega l in
           isort l'

let f4 l = let l' = append_all l in
           isort l'

let f5 l = append (append (l,[]), l)
```

a) Manually derive an upper bound on the number of ticks that are executed by each of these functions.

b) Determine if you can derive an upper bound on the number of ticks for each function using the type system for linear amortized analysis that is defined in Figure 1. If you are able to derive a bound then provide the annotated type of the respective functions and all functions that are used in the type derivation. If you are not able to derive a bound with the type rules then explain with a couple of sentences why the program is not typeable.

7.3 (18 Points) Sum Types and Amortized Analysis

I this problem, we will extend our linear amortized analysis by adding sum types $T_1 + T_2$ to our language. To this end we define data types as.

$$T \quad ::= \quad X \mid \mathrm{L}(T) \mid \cdots \mid T_1 + T_2$$

The syntax is extended as follows:

$$\begin{array}{rcl} e & ::= & x \\ & & \mathrm{app}(f, x) \\ & & \mathrm{matL}(x, e_1, (x_1, x_2).e_2) \\ & & \ldots \\ & & \mathrm{inl}(x) \\ & & \mathrm{inr}(x) \\ & & \mathrm{matchS}(x, x_1.e_1, x_2.e_2) \end{array}$$

We add the following type rules to the affine type system for base types.

$$\frac{}{x : T_1 \vdash^{\ell} \mathrm{inl}(x) : T_1 + T_2}\ (\text{T:InL}) \qquad \frac{}{x : T_2 \vdash^{\ell} \mathrm{inr}(x) : T_1 + T_2}\ (\text{T:InR})$$

$$\frac{\Gamma, x_1 : T_1 \vdash^{\ell} e_1 : T \qquad \Gamma, x_2 : T_2 \vdash^{\ell} e_2 : T}{\Gamma, x : T_1 + T_2 \vdash^{\ell} \mathrm{matchS}(x, x_1.e_1, x_2.e_2) : T}\ (\text{T:MatSL})$$

Values are defined as follows.

$$v ::= \ell \mid (\ell_1, \ell_2) \mid \text{true} \mid \text{false} \mid \text{inl}(\ell) \mid \text{inr}(\ell)$$

Finally, we add the following evaluation rules to big-step semantics.

$$\frac{H' = H, \ell \mapsto \text{inl}(V(x))}{V; H \vdash \text{inl}(x) \Downarrow (\ell, H') \mid M^{\text{inl}}} \ (\text{EE:INL}) \qquad \frac{H' = H, \ell \mapsto \text{inr}(V(x))}{V; H \vdash \text{inr}(x) \Downarrow (\ell, H') \mid M^{\text{inr}}} \ (\text{EE:INR})$$

$$\frac{V(x) = \text{inl}(\ell) \qquad V[x_1 \mapsto \ell]; H \vdash e_1 \Downarrow (\ell', H') \mid (q_1, q_2)}{V; H \vdash \text{matchS}(x, x_1.e_1, x_2.e_2) \Downarrow (\ell', H') \mid M^{\text{matS}} \cdot (q_1, q_2)} \ (\text{EE:MATSL1})$$

$$\frac{V(x) = \text{inr}(\ell) \qquad V[x_2 \mapsto \ell]; H \vdash e_2 \Downarrow (\ell', H') \mid (q_1, q_2)}{V; H \vdash \text{matchS}(x, x_1.e_1, x_2.e_2) \Downarrow (\ell', H') \mid M^{\text{matS}} \cdot (q_1, q_2)} \ (\text{EE:MATSL2})$$

a) Extend the definitions of linear annotated types and the definition of potential functions to sum types $T_1 + T_2$.

b) Define linear resource-annotated type rules for the new syntactic forms for sum types.

c) Show that your rules are sound with respect to the big step semantics. Remember that the soundness theorem of amortized analysis is proved by a nested induction on the evaluation judgement and the type derivation. Consider the cases in which the derivation type judgement ends with one of the new type rules for sum types.

$\Sigma;\Gamma \vdash^{q}_{q'} e : B$	Given resource metric M, expression e has annotated type A under signature Σ in context Γ.

$$\frac{q = q' + M^{\mathsf{var}}}{\Sigma; x : B \vdash^{q}_{q'} x : B}\ \text{(L:VAR)} \qquad \frac{A \xrightarrow{p/p'} B \in \Sigma(f) \qquad q = p + M^{\mathsf{app}}}{\Sigma; x{:}A \vdash^{q}_{p'} \mathrm{app}(f, x) : B}\ \text{(L:APP)}$$

$$\frac{\Sigma;\Gamma_1 \vdash^{p}_{p'} e_1 : A \qquad \Sigma;\Gamma_2, x{:}A \vdash^{p'}_{q'} e_2 : B \qquad q = p + M^{\mathsf{let}}}{\Sigma;\Gamma_1,\Gamma_2 \vdash^{q}_{q'} \mathrm{let}(e_1, x.e_2) : B}\ \text{(L:LET)}$$

$$\frac{e \in \{\mathrm{true}, \mathrm{false}\} \qquad q = M^{\mathsf{cbool}} + q'}{\Sigma;\cdot \vdash^{q}_{q'} b : \mathrm{Bool}}\ \text{(L:BCONST)}$$

$$\frac{\Sigma;\Gamma \vdash^{q_1}_{q'} e_1 : B \qquad \Sigma;\Gamma \vdash^{q_2}_{q} e_2 : B \qquad q = M_1^{\mathsf{cond}} + q_1 \qquad q = M_2^{\mathsf{cond}} + q_2}{\Sigma;\Gamma, x : \mathrm{Bool} \vdash^{q}_{q'} \mathrm{if}(x, e_1, e_2) : B}\ \text{(L:COND)}$$

$$\frac{q = M^{\mathsf{pair}} + q'}{\Sigma; x_1 : A_1, x_2 : A_2 \vdash^{q}_{q'} \mathrm{pair}(x_1, x_2) : A_1 * A_2}\ \text{(L:PAIR)}$$

$$\frac{\Sigma;\Gamma, x_1{:}A_1, x_2{:}A_2 \vdash^{p}_{q'} e' : B \qquad q = M^{\mathsf{MatP}} + p}{\Sigma;\Gamma, x : A_1 * A_2 \vdash^{q}_{q'} \mathrm{matP}(e, (x_1, x_2).e') : B}\ \text{(L:MATP)} \qquad \frac{q = M^{\mathsf{nil}} + q'}{\Sigma;\cdot \vdash^{q}_{q'} \mathrm{nil} : L^p(A)}\ \text{(L:NIL)}$$

$$\frac{q = M^{\mathsf{cons}} + p + q'}{\Sigma; x_1 : A, x_2 : L^p(A) \vdash^{q}_{q'} \mathrm{cons}(x_1, x_2) : L^p(A)}\ \text{(L:CONS)}$$

$$\frac{\begin{array}{c}\Sigma;\Gamma \vdash^{q_1}_{q'} e_1 : B \\ \Sigma;\Gamma, x_1 : A, x_2 : L^p(A) \vdash^{q_2}_{q'} e_2 : B \qquad q = M_1^{\mathsf{matL}} + q_1 \qquad q + p = M_2^{\mathsf{matL}} + q_2\end{array}}{\Sigma;\Gamma, x{:}L^p(A) \vdash^{q}_{q'} \mathrm{matL}(x, e_1, (x_1, x_2).e_2) : B}\ \text{(L:MATL)}$$

$$\frac{\Sigma;\Gamma, x_1 : A_1, x_2 : A_2 \vdash^{q}_{q'} e : B \qquad A \curlyvee (A_1, A_2)}{\Sigma;\Gamma, x : A \vdash^{q}_{q'} \mathrm{share}(x, (x_1, x_2).e) : B}\ \text{(L:SHARE)}$$

$$\frac{\Sigma;\Gamma, x{:}A \vdash^{q}_{q'} e : B \qquad A' <: A}{\Sigma;\Gamma, x{:}A' \vdash^{q}_{q'} e : B}\ \text{(L:SUPERTYPE)} \qquad \frac{\Sigma;\Gamma \vdash^{q}_{q'} e : B \qquad B <: B'}{\Sigma;\Gamma \vdash^{q}_{q'} e : B'}\ \text{(L:SUBTYPE)}$$

$$\frac{\Sigma;\Gamma \vdash^{p}_{p'} e : B \qquad q \geq p \qquad q - p \geq q' - p'}{\Sigma;\Gamma \vdash^{q}_{q'} e : B}\ \text{(L:RELAX)} \qquad \frac{\Sigma;\Gamma \vdash^{q}_{q'} e : B}{\Sigma;\Gamma, x{:}A \vdash^{q}_{q'} e : B}\ \text{(L:WEAK)}$$

Figure 1: Linear resource-annotated type rules.