Assignment 4:
Automatic Amortized Resource Analysis

15-819: Foundations of Quantitative Program Analysis (Fall 2019)

Out: Tuesday, October 22, 2019
Due: Tuesday, November 05, 2019 11:59pm EDT

1 Project Proposal

Important: Briefly discuss your project idea with the instructor before writing the project proposal.

In this task you write a proposal for a final project. *A course project can be a short paper or an implementation. You can complete the project individually or as a team of two.* In any case, you will present your project in a 10 minute presentation during the last lecture on December 5. Your project should complement the material that has been discussed in the course. The project needs to be completed before

December 3, 2019.

Below are some suggestions for course projects. However, you are encouraged to propose your own ideas for projects. It is good to be ambitious. However, keep time management in mind. The total time you spend with the project should be similar to the time you have spent with 2 of the bi-weekly assignments so far.

a) Identify an interesting application area for resource bounds and implement a case study in Resource Aware ML

b) Implement a simple automatic solver for recurrence relations

c) Implement (or extend) System SLR

d) Implement (or extend) System BC

e) Write a paper about the proof that System SLR corresponds to the non-size-increasing functions in the complexity class FP

f) Write a paper about a soundness proof for linear AARA that uses a structural operational dynamics.

g) Write a paper about extending linear AARA so that functions can be treated both in an affine or unrestricted way

h) Proof an interesting theorem that we discussed in the course with a proof assistant

i) Write a paper about a related research result that we did not discuss

Task 1.1 (15 pts). Start writing the proposal *after* you agreed with the instructor on a topic. Your proposal should be roughly *1-page long* and include the following parts.

1) Title and project type (short paper, or implementation)
2) Motivation (Why are you interested in this project? How does it relate to the topics discussed in class?)
3) Background (Summarize the existing technical results that are most relevant for your project.)
4) Project description (Outline the scope and goals of your project, e.g., a sketch of the short paper.)
5) Milestones and timeline (Define at least three milestones with dates that you need to complete for a successful project)

2 Expressivity of Linear AARA

In this problem, we study example functions that illustrate the expressivity of linear automatic amortized resource analysis (AARA). As in lecture, we use *tick* expressions to define the resource usage. The following examples use the syntax of Resource Aware ML (RAML), which is a sub-language of OCaml. RaML is an implementation of *multivariate polynomial amortized resource analysis* which is more expressive than linear AARA. For example, RaML supports curried functions like append for lists, bounds that are function of user-defined inductive types, let polymorphism, and multivariate polynomial bounds like $n{\cdot}m$. A web interface for RaML is available at

`http://raml.co`

The RaML website also contains installation and usage instructions.

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

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

let rec omega : int list → int list =
  fun x → omega x
```

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

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

let f1 (x,y) = repeat (x, append_all y)

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' = f1 l in
           append_all l'

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

The functions use integers, which we did not introduce in lecture. They are treated exactly like the unit value. The type rule for integer constants is given below.

$$\frac{n \in \mathbb{Z}}{\cdot;0 \vdash n : \langle n, 0\rangle} \text{ (L:INT)}$$

Task 2.1 (11 pts). Manually derive a tight upper bound on the number of ticks that are executed by each of these functions, that is, precisely describe the worst-case number of ticks executed as a function of the sizes of the inputs.

Hint: You can use RaML to derive worst-case bounds. Note however that RaML does not necessary derive a precise bound. It only guaranties that the bound is sound.

Task 2.2 (22 pts). Determine for each of the functions if you can derive a typing using the type system for *linear* AARA from lecture. If you are able to derive a bound then provide the annotated type of the respective function 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.

3 Univariate Polynomial AARA

Recall the typing rules of *univariate polynomial* automatic amortized resource analysis from lecture.

Consider the implementation of quick sort that is given below. To derive a bound on the number of comparison operations that are performed during an evaluation of the function *quicksort*, we inserted tick expressions at each place in the program at which we perform a comparison.

```
let rec append : int list * int list → int list =
  fun (l1,l2) →
  match l1 with
  | [] → l2
  | x::xs → x::(append (xs,l2))

let rec partition : int * int list → int list * int list =
  fun (y,l) →
  match l with
  | [] → ([],[])
  | x::xs →
     let (cs,bs) = partition (y,xs) in
     let _ = Raml.tick 1.0 in
     if y<x then
       (cs,x::bs)
     else
       (x::cs,bs)

let rec quicksort l =
  match l with
  | [] → []
  | x::xs →
     let (ys,zs) = partition (x,xs) in
     append (quicksort ys, x :: (quicksort zs))
```

Task 3.1 (10 pts). Provide resource annotated types for the functions *append*, *partition*, and *quicksort* that are derivable in univariate polynomial AARA.

Task 3.2 (10 pts). Give a type derivation for the resource annotated type of *quicksort* that you provided in the previous task. (You don't have to provide type derivations for *partition* and *append*.)

4 Solving Recurrence Relations with RaML

In this problem, you use RaML to encode and solve the recurrences. Encoding a recurrence in RaML means to create a program that has the (worst-case) resource cost that is defined by the recurrence.

Recall the following recurrence relations from Assignment 1.

a) $R(m,n) = 0$ for $n < 4$ and $R(m,n) = R(m,n-4) + 12(m + \binom{m}{2})$ for $n \geq 4$

b) $T_1(0) = T_2(0) = 0$, $T_1(n) = T_2(n-1) + 20(n-1)$, and $T_2(n) = T_1(n-1) + 8n - 2$ for $n > 0$

c) $S(1) = 0$, $S(2) = 1$, and $S(n) = S(\lceil n/2 \rceil) + S(\lfloor n/2 \rfloor) + n - 1$ for $n > 0$

Task 4.1 (15 pts). For each of the recurrences, define a RaML function that encodes the recurrence.

Task 4.2 (6 pts). For each of the encoded functions from the previous task: Derive a bound with RaML, report the bound, and compare it to your manually derived bound.

Hint: The recurrence $T(0) = 10$ and $T(n) = T(n-1) + 3$ for $n > 0$ can be encoded by the following program.

```
let rec t l =
  match l with
  | [] → Raml.tick(10.0)
  | x::xs → let _ = Raml.tick(3.0) in t xs
```

5 Typos in Lecture Notes

Task 5.1 (10 pts). You get 2 bonus points for every typo you find in the lecture notes published after October 1 (“From System T to Systme BC” or later). You can not get more than 10 bonus points per assignment.