RaML: Automatic Resource Bound Analysis
15-300 Research Proposal Summary

Michael Zhang
November 1, 2019

1 Web Page
http://andrew.cmu.edu/user/jinyaoz

2 Project Description

The automatic determination of the quantitative resource consumption of programs is a classic research topic which has many applications in software development. Recently, a multivariate amortized resource analysis that automatically computes polynomial resource bounds was developed by a group of researchers led by Prof. Jan Hoffmann from Carnegie Mellon University. [3] The language that implements this analysis is called Resource Aware ML (RaML), which is a functional language that was integrated over OCaml. [2] It automatically and statically computes the upper bounds of the worst-case resource use and the lower bounds of the best-case resource use for a program where the resource could be the number of evaluation steps or even user-defined ticks. It supports higher-order, polymorphic programs as well as side effects and user-defined inductive types.

This project involves expanding the implementations of RaML and improving the user-interface of RaML. Currently, RaML does not fully include all the expressions in the standard library of OCaml. For example, labels and modules in OCaml are not yet implemented in RaML. While deriving the cost analysis in modules is not yet known to be possible, expanding the language with labelled tuples is an achievable goal and would greatly improve the user-friendliness of RaML with explicit mappings of expressions to their labels in sets and tuples. Meanwhile, we also aim to improve the interface of RaML, making it more explicit with its resource inference points.

3 Project Goals

The overall goal of the project is to expand the implementations of RaML to support more built-in Ocaml expressions and improve the interface of RaML.

• 75% Goal

Implementing Labels in RaML. Define the statics and dynamics of Labels in RaML, and develop the amortized analysis algorithm for labels. After proving the soundness and completeness of the definition, we will implement the LP solver corresponding to labels, and formally include labels in RaML.
• 100% Goal
After expanding the implementations of RaML, we will work on improving the interface of RaML. Currently, the user interface of RaML consists of the inferred type and computed worst-case resource bound for the program at compile time. We want to make the interface more specific with line specifications of the causation of the costs. For instance, if a program consumes $O(n)$ clock ticks, the improved interface would be able to point out the line where the cost was caused. We would like to start this improvement with simple inductive datatypes such as list.

• 125% Goal
Developing existential types in RaML. To include modules in RaML, one has to first develop the typing rules of pack and open, which involves defining existential types and built statics and dynamics for them. In this project, we would start to develop the dynamics of modules, and design the run-time behavior of RaML when the types in a module are unveiled. This will pave the way for introducing modules in the language in the future.

4 MileStones
In this section, we will propose one milestone towards the end of 15-300 and four monthly milestones for 15-400 as detailed biweekly milestones are not ready to be proposed yet.

4.1 15-300 Milestone
Complete a survey of the statics and dynamics of RaML. Understand the abstract machine model for RaML.

4.2 15-400 Milestones
1. Learn the LP solver of RaML. Study the cases of the behavior of the amortized LP solver for arbitrary inductive datatypes.
2. Define the types, expressions, statics, and dynamics of labels in RaML. Define the abstract stack machine evaluation model for labels.
3. Implement the LP solver for labels. Improve the interface of RaML with details discussed in the previous section.
4. Develop the typing rules for signatures and modules (existential types).

5 Literature Search
The project of RaML started in 2012, when Jan Hoffmann, Klaus Aehlig, and Martin Hofmann published the tool paper of the prototype of Resource Aware ML [3]. The paper proposed the first type based resource analysis system, with a syntax-directed statics analysis that derives inequalities over the resource bound. The first version of RaML was implemented with Haskell, and it only supports first order functions.

In 2015, a more formal discussion of type-based amortized resource analysis system with integers and arrays was presented by Jan Hoffmann and Zhong Shao [1]. The paper proposed a more
developed resource aware type system, where the typing rules encodes evaluation information. This paper was the theoretical foundation of RaML.

RaML has been developed in multiple directions in recent years. For example, it was extended with a novel type system that verifies programs that consumes constant resource [4]. In 2017, RaML was extended that it supports user-defined recursive datatypes [2]. This was a huge step towards the goal of providing automatic resource analysis for the full version of OCaml. Sum and product types were also introduced so it supports multivariate resource analysis and high order functions [2].

6 Resources Needed

This project does not require any additional hardware or software resources. We already have everything that is needed.
References


