Computer-Checked Recurrence Extraction for Functional Programs

Bowornmet Hudson, **Daniel R. Licata**
Wesleyan University

Contact Information:
Department of Mathematics and Computer Science
Wesleyan University
Middletown, CT 06459

E-mail: bhudson@wesleyan.edu

Introduction

Time complexity analysis for a recursive program can be broken down into two steps:

1. Extracting a recurrence relation that describes the running time in terms of the size of its input
2. Reducing the recurrence to obtain a closed form and asymptotic bound on the running time

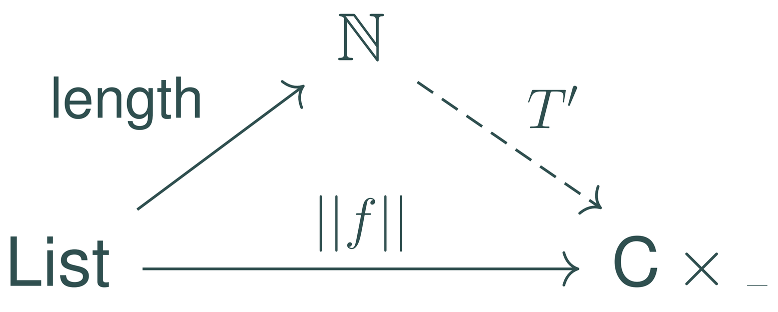
This can be an arduous process, as it is usually computed by hand; this is especially true for large, elaborate pieces of code. The overall aim of this project is to develop a formal system for automated complexity analysis in Agda, a dependently-typed programming language and proof assistant.

Thesis Statement:

It is possible to extract and formally reason about time complexity properties of functional programs using proof assistants.

Our Approach

- Following [1], we extract recurrences from source programs using a monadic translation $||\cdot||$ into a complexity language.
- The complexity language is equipped with an abstract preorder judgement $\_\leq s\_$ which specifies ordering on terms and allows us to manipulate recurrences into closed forms, from which we can deduce asymptotic bounds on the running time of the original source program.
- We implement and prove properties about our system in a proof assistant, Agda, to ensure the correctness of the language specifications and cost information extracted by our system.



Source Language

```
data Tp : Set where
 unit : Tp
 nat : Tp
 susp : Tp → Tp
 _->s_ : Tp → Tp → Tp
 _×s_ : Tp → Tp → Tp
 list : Tp → Tp
 bool : Tp
```

Complexity Language

The complexity language is designed to be a setting where we can reason about program costs directly without needing to appeal to a denotational semantics. The abstract cost measures specified by the source language operational semantics are interpreted in the complexity language as cost values of type C.

Translation

The translation function $||\cdot||$ from source to complexity is the main mechanism for recurrence extraction in our system. The translation of a source term $e : \tau$ returns a cost-potential pair of type $\mathsf{C} \times \langle\langle\tau\rangle\rangle$, where the potential captures the notion of the upper bound on the size of a term. We prove that the results obtained by the translation are an upper bound on the costs specified by the source language operational semantics.

Denotational Semantics

We interpret the types and terms of the complexity language as preorders and monotone functions between preorders.

```
record Preorder-str (A : Set) : Set1 where
 constructor preorder
 field
 _ ≤ _ : A → A → Set
 refl : ∀ x → x ≤ x
 trans : ∀ x y z → x ≤ y → y ≤ z → x ≤ z
```

Future Work

1. Study methods for solving complexity language recurrences into closed forms, from which we can deduce an asymptotic bound on the running time of source programs.

References

[1] Norman Danner, Jennifer Paykin, and James S. Royer. A static cost analysis for a higher-order language. In Proceedings of the 7th Workshop on Programming Languages Meets Program Verification, PLPV 13, pages 2534, New York, NY, USA, 2013. ACM.