This document provides a brief summary of work done with Joshua Liebow-Feeser as the final project for CSCI-1260 (Compilers).

We designed a small proof-of-concept programming language, Sb, with an implementation that synthesizes runtime estimates for programs. Sb is implemented as a domain-specific language in Idris [2], a functional language with *dependent types*. We developed the following key design criteria:

(1) *Expressiveness*. Sb should allow the programmer to express nontrivial programs.

(2) *Accuracy*. Sb should provide accurate estimates of program time complexity.

(3) *Automation*. Sb should allow as many proof obligations as possible to be discharged without the intervention of the programmer.

Expressiveness Following the approach outlined by Meijer et. al. in [3], our language has basic built-in combinators (e.g. `fold`) that are sufficient for expressing a number of common programming tasks. For programs that lie outside the scope of these combinators we provide a *cost monad* that allows programmers to express runtime information about arbitrary Idris programs, and then lift those runtime costs into Sb.

Automation and Accuracy Our initial approach to analyzing Sb programs was to develop a big-step operational semantics of the language (expressed as a dependent type in Idris, similar to the approach by Pierce et. al. in [4]), along with a *cost model* in the sense of Blelloch et. al. in [1]. While this approach yielded a completely accurate description of program complexity, it created additional proof obligations to verify the runtime of simple programs — violating our automation requirement. We instead developed the notion of a *cost-indexed abstract syntax tree* that associates every syntactic Sb construct with a cost, derived in terms of the cost of its children. This allows for simple algorithms to be expressed as macros: functions on the Sb syntax trees that express maximum blowup in cost. While this latter approach is imprecise (e.g. it provides inexact answers for conditional branches), it provides cost estimates in a completely automated fashion.

References

[1] Guy Blelloch and John Greiner. Parallelism in sequential functional languages. In *Proceedings of the seventh international conference on Functional programming languages and computer architecture*, pages 226–237. ACM, 1995.

[2] Edwin Brady. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming*, 23(05):552–593, 2013.

[3] Erik Meijer, Maarten Fokkinga, and Ross Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *Functional Programming Languages and Computer Architecture*, pages 124–144. Springer, 1991.

[4] Benjamin C. Pierce, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjoberg, and Brent Yorgey. *Software Foundations*. Electronic textbook, 2015. `http://www.cis.upenn.edu/~bcpierce/sf`.