Project Proposal
CSCI P523
Prof. Jeremy G. Siek
AI: Michael Vitousek

Hindley Milner Type Inferencer, Compiler Extensions and Optimizations
Teammates: Yang Zhang, Srikanth Kanuri
Proposal Date: 04/13/2016

Problem Statement

As part of Assignments, we have built up a compiler for the subset of Racket with three intermediate languages along with several features and optimizations like Register Allocation, Move Biasing, Garbage Collection, etc.[1] As a part our final project, we would like to extend this language by implementing lists and characters and also a few interesting optimizations like function inlining[2], Argument Biasing and Constant Folding. We also intend to implement a standard type inference system i.e. The Hindley Milner Type Inference Algorithm[3, 4] to infer types and produce a type annotated expression for the input. The resultant compiler is expected to have a new type inferencer which generated types for the input, better optimizations and also compiles a more comprehensive language.

Analysis & Solution

There a several things we wanted to implement and we will analyze each of them below.

The Hindley Milner Algorithm[3, 4] will have two main areas
1) Constraint Generation
2) Constraint Solving and Unification.
Constrain generation will deal with generating type variables for the normal variables in code and generate constraints on them. The second part will be dealing with solving the constraints by unification. The input to the compiler will no longer need types to the lambda or functions after the type inferencer. The inferencer will annotate the types and give out the output so it can be used in the following passes.

We intend to implement the list like the box and arrow implementation in Scheme. Each list will be implemented as a list of vectors. A Single element will be present in a vector and the reference to the next element will be stored in the vector.

Argument Biasing is similar to Move Biasing but it needs to identify the values which are more probable to be sent as parameters to the function and use the respective register to assign that value. Constant folding is an optimization to simplify the arithmetic operations which include constants but replacing the expression with the result of the expression.

Function Inlining[2] is a more involved optimization where we bring in the function code into the caller's code space. Function Inlining usually increases the code space. But if properly used, it reduces the overhead of having a new stack block, register moves, etc. We need to do an analysis first to identify if inlining is advantageous before actually implementing it.

Timelines

We propose to complete a basic working prototype with these features in two weeks. In this first week, we will implement list and character features and with constraint generation in the typechecker. In the second week, we will try to implement the optimization features and the unification in typecheker. We will try to implement a simpler version of function inlining since the actual function inlining is more complicated, needs more changes and is difficult to accomplish in the the given time. Since we have not finished Assignment-7 yet, we will need some additional time to finish a substantial prototype of what we propose.

Future Work

The Static Symbol Assignment form[6] promises an optimized Register Allocation[5] and also a chance for implementing better optimizations like Constant, Copy Propagation, Dead Code Elimination, and other Biasing forms. Hence, we would like to add SSA form as an intermediate language after the Pseudo C Language. We also intend to implement a complete version of function inlining as mentioned in the paper "Fast and Effective Procedure Inlining"[2]

References

[1] Jeremy G. Siek. *Essentials of Compilation - An Incremental Approach*

[2] Oscar Waddell and R. Kent Dybvig. *Fast and Effective Procedure Inlining.*

[3] Bastiaan Heeren, Jurriaan Hage and Doaitse Swierstra. *Generalizing Hindley-Milner Type Inference Algorithms*

[4] Martin Grabmuller. *Algorithm W Step by Step*

[5] Sebastian Hack, Daniel Grund, and Gerhard Goos. *Register allocation for programs in SSA-form.*

[6] Marc M. Brandis and Hanspeter Mosenbock. *Single-Pass Generation of Static Single-Assignment Form for Structured Languages*