CS 480, Senior Project Proposal
Kurt Kilpela
Fall 2014

Objective

To research and implement various optimizations for Modtalk code generation.

Project Details

As with any compiler that schedules tiles of code, operations can become repetitive. Overlap may exist between these tiles of code resulting in multiple operations with no net benefit. The first optimization I'll explore will relate to removing this overlap via a Peephole optimizer. Secondly, I will begin to explore control flow and SSA optimizations.

With polymorphism, message send call sites can be monomorphic (one receiver type), polymorphic (multiple receiver types) or megamorphic (too many receiver types). With the case of monomorphic call sites, we can bind the method to call and do a quick check at the beginning of the method to ensure the receiver is the correct type. With a polymorphic call site, we can include in the code a sequence of receiver type checks and branch instructions such that we can find the method faster than doing a lookup. The site could be re-written dynamically or statistics could be collected during a run to ensure the order of the type checks will provide the fastest lookup. Megamorphic call sites have too many receiver types, so finding megamorphic call sites and ensuring they do a standard lookup will result in faster code.

Method without a message send are called leaf methods. For each method, we build a frame. Building a frame is unnecessary, so with leaf methods, we can apply optimizations that remove about 80% of the instructions in our leaf methods. Another place we can remove code is when the code is dead. Dead code elimination is a size optimization.

If time remains, other optimizations will be explored.

Grading

Optimization	Points
Peephole optimizer	10
Leaf method optimization	10
Message Send Caching	30
CFG	30
SSA Form and Phi assignment	30
Control Flow Optimizations	30
Dead code elimination	10

Total is 150

Grade Scale

Number of points	Grade
130	A
110	B
90	C
70	D
<70	F