CS401 Optimization Week 1 report .

Starting the week, I read chapter 11 of Mogensen's Basics of Compiler design. After reading the chapter, I looked through the Mil java code and I was able to recognize the code analysis that was presented in the chapter. I ran through the included mini code examples, and created modified versions to compare the output. For example, modifying arrs.mil to use a function to set the value of N, the resulting Mil generated code is almost the same as the original, with an added starting block, that immediately calls the original starting block. I noted that the value of N in the block b0 was not stored, and was moved to the call to block 7.

```
  not recursive
b0() =
    a <- newarray((40))
    _ <- store((a, 0))
    t22 <- add((a, 4))
    _ <- store((t22, 1))
    b7(10, 2, a)
```

From my reading in Morgensen, it initially seemed like the variable N should no longer be live after the store of 0,1 to the array, but after some consideration, it seemed that it may be because aliasing cannot occur without a function call in mini, but I was not certain.

Thinking about the global optimization for replacing N with 10 in this code, I developed an algorithm that I believe will be suitable. Because I am not yet familiar with the optimizer, I was unable to add this optimization.

Algorithm:

Run static analysis on the block, at the invocation of a new block, compute the set difference of the block dependencies and the live variables, if this set is not empty, check the callees of the block to see if a suitable wrapper block exists, if not create a wrapper block that calls the original block, but only depends on the set difference, and uses the constant from before, otherwise use the previous wrapper block. Use a predetermined maximum value for wrappers to a particular block, to prevent code explosion.

If the only callers to a block are wrapper blocks, and the number of wrapper blocks is less than the max inline the original block.