1 Value Numbering

At a very basic level, most optimization techniques revolve around avoiding redundant or unnecessary computation. Thus, it's vital that we discover which values in a program are equal. That way, we can simplify the code that wastes machine cycles repeatedly calculating the same values. Classic optimization phases like constant/copy propagation, common subexpression elimination, loop-invariant code motion, induction variable elimination, and others discussed in the de facto treatise, "The Dragon Book", perform this sort of redundancy elimination based on information about the equality of expressions.

cite

In general, the problem of determining whether two expressions in a program are equivalent is undecidable. Therefore, we seek a *conservative* solution that doesn't necessarily identify all equivalences, but is nevertheless correct about any equivalences it does identify. Solving this equivalence problem is the work of *value numbering* algorithms. These assign every value in the program a number such that two values have the same value number if and only if the compiler can prove they will be equal at runtime.

Value numbering has a long history in literature and practice, spanning many techniques. In **??** we saw the `value-numbering` word, which is actually based on some of the earliest—and least effective—methods of value numbering. **??** describes the way Factor's current algorithm works, and highlights its shortcomings to motivate the main work of this thesis, which is covered in Sections 1 and 1.1. We finish the section by analyzing the results of these changes and reviewing the literature for further enhancements that can be made to this optimization pass.

1.1 avail