Program Dependence Graphs

1 The Program Dependence Graph and Its Use In Optimization

```
@article{ ref:PDG1,
 author={Jeane Ferrante and Karl J. Ottenstein and Joe D. Warren},
 year={2987},
 title={The Program Dependence Graph and Its Use In Optimization},
 journal={ACM Transactions on Programming Languages and Systems (TOPLAS)},
 volume=9,
 issues=6,
 pages={319--349}
}
```

Summary: This paper introduces the notion of a Control Dependence Graph (CDG), which encodes which parts of a program fragment can control which other parts of a program execute or not. The CDG can be combined with pre-existing Data Dependence Graphs (DDG) to form the Program Dependence Graph (PDG). The PDG is useful in a compiler, to prove which blocks of code can be moved where when optimizing code (e.g. justifying Code Motion).

Evaluation: PDG's are still used in modern compilers. Thus this paper sets the foundations for three decades of compiler research. The core structure of the PDG layed out here is surprisingly simple (once the slightly twisty definitions of domination and dependence are internalized). Whats more important is that the a PDG is simple to use, and using one doesn't require internalizing the twisty definitions. However PDG generation does have to be tuned to the language in question, and this is little discussed in the paper. For example: the handling of loops in PDG's as presented is unsound in any language where loop termination isn't somehow guaranteed (C/C++ for example). Much of the work in the last few decades has been about adapting PDG's to different languages (such as languages with explicit concurrency features, or languages that don't guarantee loop termination). While this could be viewed as a flaw of this work, it also goes to show just how foundational and adaptable PDGs are as an intermediate program representation for may kinds of languages.

2 Efficiently Computing Single Static Assignment Form and the Control Dependence Graph

```
@article{ ref:PDG2,
 author={Ron Cytron, Jeanne Ferrante, Barry K. Rosen, F. Kenneth Zadek},
 title={Efficiently Computing Single Static Assignment Form and the
Control Dependence Graph},
 year={1991},
 journal={ACM Transactions on Programming Languages and Systems (TOPLAS)},
 volume=13,
 issue=4,
 pages={451--490}
}
```

Summary: This paper introduces the notion of a Dominance Frontier (DF). Using this notion it gives a linear time (in number of edges and size of the DF) algorithm for computing CDG's, and a separate algorithm for computing SSA form in "good" time. (The actual runtime is complicated, but remains mostly linear in the size of the graph, DF, and number of variable assignments).

Evaluation: This method of computing the CDG is still used today. And, while this paper does not introduce SSA, having a more efficient method of computing it makes it viable, which is important since SSA is often used as the core IR of many compilers (like LLVM). In addition, the notion of the DF is clearly powerful as it allows the CDG and the algorithm for computing it to be defined in a single sentence: "... Control Dependencies are essentially the dominace frontiers of the *reverse* graph of the control flow graph".