Name: Kathryn S McKinley **Date:** September 26, 2009.

Paper: (Copy the citation from the webpage). Backus, Beeber, Best, Goldberg, Haibt, Herrick, Nelson, Sayre, Sheridan, Stern, Ziller, Hughes, and Nutt,"The Fortran Automatic Coding System" Proceedings of the Western Joint Computer Conference, pp. 187-198, Los Angeles, CA, February, 1957.

Summary: *(The summary should contain a short description of the problem, solution, and meaning of the paper. For example, for this paper you might write the following.)*

(Problem) In the 1950s, computer languages and architectures were in a very primitive state. In particular, there were no high-level languages, only assembly languages. (Solution) Backus and his colleagues introduced the first programming language (Fortran) and its compiler for translating from this high-level language to machine language. (Meaning) Before these inventions, computers could only be programmed using assembly code. By inventing the first high-level language and its compiler, this paper changed the landscape of computing by offering a much better way for people to interact with computers.

Strengths: (One to three sentences on strengths of the paper.) The paper shows for the first time how to translate from high-level Fortran (a language that uses procedures, loops, arrays, and scalar variables) to an intermediate form, perform optimizations, and then translate to assembly code. It invented control-flow graphs and the first redundancy elimination optimizations.

Weaknesses: (One to three sentences on weaknesses of the paper.) The paper's weaknesses are that it does not discuss or prove why or if it is always possible to perform this translation. The paper does not present any experiments to explain the effectiveness of the approach.

Analysis I: (Discuss in detail some interesting aspect of the paper in detail.)

This paper totally changed the field of computing. In this era, hardware was extremely expensive and only a few people had access or the ability to program it. This paper set the stage for dramatically improving the productivity of programmers. All high-level programming evolved from this starting point, and this paper started the programming language implementation field (i.e., compilers, interpreters, etc.). It is truly amazing how much compiler technology was invented and predicted by this single paper, e.g., basic blocks, frequency profiling, and high-level intermediate representations.

For example, this paper introduces the *separation of concerns* principle. Consider register allocation. The compiler assumes an infinite temporary register set during the rest of compilation, which simplifies the rest of the compiler. It does not have to consider how optimizations interact with register allocation early in compilation. In their system, towards the end of compilation, the compiler assigns the 3 registers of the 701 based on execution frequency from profiling. It computes basic block live ranges and finds interferences on the fly. This powerful abstraction is very useful for ignoring resource constraints and simplifying other parts of the compiler. There are some places where it however breaks down, such as embedded, VLIW, EDGE, and other

architectures which have more than one resource constraint. For example, VLIW machines must carefully deal with both register allocation and instruction scheduling in a fixed width instruction. Which resource do you do first? And how does it constrain the others? This problem remains an important one in the literature for many situations, and the only solutions, of which I am aware, have been iteration [1, 2], which is not completely satisfying.

(Aside: Brasier et al. preset a solution to register allocation and scheduling [1]. They iterate between scheduling, which may add register pressure, and register allocation. If the schedule does not require any *spilling* (spilling means a value comes from loads and stores instead of a register access), they are done, otherwise, they constrain the schedule to limit increases to live ranges and thus obtain a register allocation with less or no spills.)

Analysis II: (Discuss a second aspect of the paper.) This paper did not include any experimental results. A very interesting experiment would have been to assign *K/2* problems to *N* programmers, half performed in assembly and half in Fortran, and then assigned an additional *K/2* problems and switch the programmers. Since each programmer does 1/2 in each language, this controls a bit for individual variation. Then, the researchers could measure programming time (productivity), i.e., the time to a correct solution in the two languages. The researchers could also measure execution time, comparing how fast the resulting programs execute, to determine the differences if any. One should also use different problem sizes, and compare if any differences are correlated to program size. Writing a short program that solves a small problem in assembly should be is easier than writing a long program that solves a harder problem. Other interesting experiments would include how much each of the optimizations improved performance, to show the impact of register allocation, common subexpression elimination, etc.

Other Advice.

Analysis Topics. Below are some suggested topics to write about, but do not feel constrained by this list.

- Describe an experiment that would help explain/explore the results better.
- How did it impact the field?
- What questions remain open?
- What experiments are missing?
- How does it really relate the previous research?
- Future directions.
- Some examples for which it will or will not work.
- Could a similar paper be published today?
- Ideas or thoughts it provoked.
- Other interesting commentary.

Hints on latex. To make this document in to a PostScript file perform the following Unix commands:

```
pdflatex critique
bibtex critique
pdflatex critique
pdflatex critique
```

This sequence will produce a file called "critique.pdf" and some other auxiliary files. You can learn more about "latex" on the web `http://www.latex-project.org/` and/or buy a book on it for reference. If you prefer to use Microsoft Word or another word processor, you may use it.

Writing well. Your critiques should be clear and grammatically correct. I highly recommend you read books on writing well. I like Joseph M. Williams and his books, and in particular his book called "Style: The Basics of Clarity and Grace" [3].

Please use the active voice: For example, the following is clearer "Brasier et al./ built a compiler." than "A compiler was built." When writing about computer systems, you should identify the subject and the time (i.e., when the action takes place). For example, likely subjects include the compiler, the microarchitecture, the virtual machine, the memory manager, and the operating system. Potential times are: ahead-of-time or just-in-time dynamic compilation, design time, and runtime.

Another pet peeve of mine is to never use "this" as the subject of your sentence. For example, write: "Brasier et al. built a compiler. This compiler iterated scheduling and register allocation to solve resource constraint problems." Do not write: "A compiler was built. This iterated scheduling and register allocation to solve resource constraint problems." Qualify the "this" with a noun. Usually "this" is referring back to something in the previous sentence and you should make it explicit to which part of the sentence it refers.

References

[1] T. S. Brasier, P. H. Sweany, S. J. Beaty, and S. Carr. CRAIG: a practical framework for combining instruction scheduling and register assignment. In *Parallel Architectures and Compilation Techniques*, pages 11–18, Cyprus, Greece, June 1995.

[2] B. A. Maher, A. Smith, D. Burger, and K. S. McKinley. Merging head and tail duplication for convergent hyperblock formation. In *IEEE/ACM International Symposium on Microarchitecture*, pages 65–76, Orlando, Florida, USA, 2006.

[3] J. M. Williams. *Style: The Basics of Clarity and Grace, 2nd Edition*. Pearson Longman, 2009.