

A PSEUDOCODE COMPILER

Nick Deibel

Abstract:

In this paper, we briefly outline our preliminary work in creating a versatile and practical compiler for pseudocode. This work relies heavily on our study of the *n-monkey* problem, for which we prove some new and interesting results. We also note the potentially deadly ramifications of this compiler technology.

Motivation:

We are all familiar with pseudocode and the intellectual exercise it provides for understanding algorithms. In fact, in many introductory programming classes, we often have students write out pseudocode before they begin programming. In an inspiring event while tutoring, a student confronted this author with the question: “Why won’t my pseudocode compile?” The student had actually typed the pseudocode into a file and attempted to compile it with g++. Instead of making a mockery of the pathetic student, the author asked himself, “Yes, why won’t pseudocode compile?” Clearly, the code was correct; therefore, the compiler must be flawed.

General Design:

In designing the pseudocode compiler, we faced the issue of whether or not we should limit ourselves to a particular style of pseudocode. While this would certainly have been easier, we view it as the coward’s solution. Instead, we decided to work with any and all styles of pseudocode. We have successfully instrumented this work in English as well as other languages, including: French, Portuguese, Pig Latin, Arabic, Swahili, Australian, and American Sign Language. We admit that we were lucky with the last one since; our set of monkeys already knew the language. We are also capable of compiling most known computer languages, including Java, C++, and Scheme. For reasons we do not understand, feeding any Microsoft language into our system causes the monkeys to quit and start writing Hollywood screenplays. We do note that our work has failed so far in creating pseudocode generated by users who speak more complicated languages like Valley Girl, Male Grunting, and Demon-Possessed Rantings.

Figure 1 shows the general design of our compiler. The clear hurdle in creating our compiler was the need for a successful natural language interpreter. However, through successful use of our *n-monkey* component, we were able to take any NLP and make it successful. In fact, an empty cardboard box nearly suffices for our needs. Weights need to be added to the box, though,

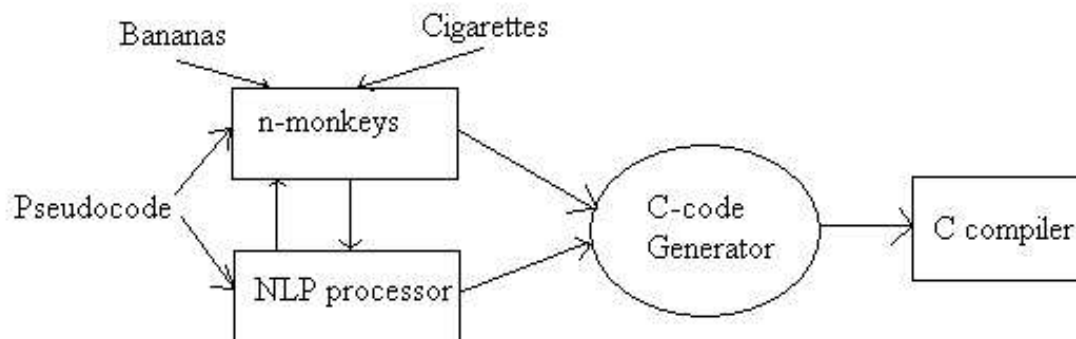


Figure 1. The general design of our pseudocode compiler.

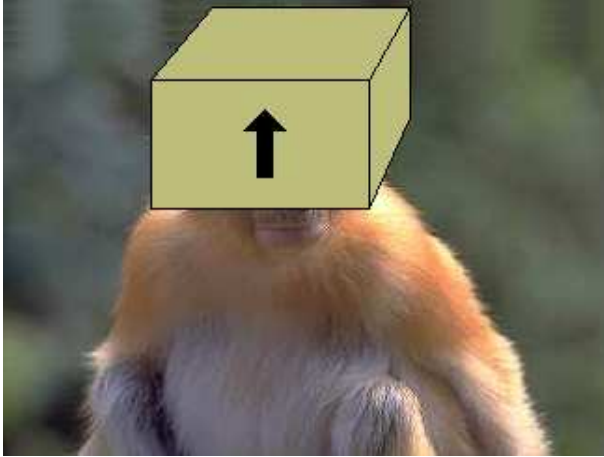


Figure 2. Monkey wearing NLP as a hat

as the monkeys are prone to using an empty box as a hat (see figure 2).

The design of the compiler is rather self-explanatory. The pseudocode is fed into both the n -monkey component and the NLP simultaneously. If either is confused, they communicate with each other. Eventually, information is sent to the code generator which produces valid C code that is then compiled.

The current compile time is a bit slow. For programs already in a “legitimate”

programming language, we experience the most difficulty. The Windows ME source took approximately 78 hours to fully compile. However, a pseudocode description of Windows ME took only 3 hours to compile and did not require Internet Explorer to be integrated into the operating system. Interesting enough, the full text of “Crime and Punishment” took 20 minutes to compile and created a music sharing system not unlike Napster.

Utilizing n -monkeys:

As we have already mentioned, the key component to the pseudocode compiler is our use of the n -monkey problem. For those unfamiliar with the problem, it was first stated formally by Dijkstra in 1987 when, after a rather funny fraternity prank on him, he found a large number of monkeys in his office. Due to his panic, he was unable to accurately count the monkeys and so reported to 911: “I’ve got n monkeys in my office! What should I do with them?”

Since that moment, computer science researchers have been trying to answer that very question. Most are familiar with the work that showed as n approached infinity, we get a good chance of the n monkeys typing out Hamlet on a typewriter. More recent work (I. M. Chimp ’99) showed that given a finite number of monkey n and bananas b , the probability of getting a specific text t increases quadratically as b increases and only linearly as n increases. We recently showed (see *Journal of Simian Computing, volume 37*) that by introducing cigarettes into the system, the

probability of getting any specific text t is given by $P(t) = \min\left(1, f(n) \cdot \frac{n}{n^2 + b^2 + cn} \cdot \left(\frac{13}{11}\right)^{c \log n}\right)$,

where $f(n)$ is the probability of feces being thrown in a room of n monkeys.

Chimp’s work has put strong bounds on the value of $f(n)$, but we have empirically improved them through careful simulation. Using the joint observation of Goodall and Notkin that the behavior of howler monkeys and graduate students has a correlation of $(1 - \epsilon)$, we placed different sized groups of graduate students into a locked room with toilet access and timed how long it took for the feces to begin flying. Due to human subjects review, we are unable to provide the exact numbers we determined.

Our compiler uses the n monkeys to stimulate the NLP whenever it gets stuck on interpreting the pseudocode. The NLP will poll the monkeys and take the most likely interpretation so far generated. It is interesting to note that independent of n , the monkeys produce usually only 3-4 interpretations. The monkeys can also inquire the NLP if they get stuck. To prevent a deadlock situation where both are stalled, we increase the flow of bananas and cigarettes. We have found that a simulated annealing approach to this flow works better than a strict hillclimbing as the latter often only produced a hill of cigarette butts and banana peels.

The Repercussions of the Pseudocode Compiler:

We have decided not to release our compiler to the public for several safety reasons. In preliminary testing, we identified threats to world security, the economy, and the mental health of computer scientists.

The threat to world security is two-fold. One, we have found that monkeys are quite literal. For example, the pseudocode "PRINT HELLO WORLD" will create not the simple program we all cherish, but will create an internet worm that will send a "hello" message to every computer it can reach. This broadcast nightmare was narrowly avoided in our lab. The second issue is that our compiler will do almost anything you ask it to. If the pseudocode contains "TRANSER ONE MILLION DOLLARS INTO MY SWISS BANK ACCOUNT," the monkeys will figure out a way to do so. Similarly deadly messages like "BLOW UP CHATANOOGA" and "MAKE THE IRISH BALD" would also likely result in programs capable of doing just that.

Even if we solve these security issues, this compiler would put millions of programmers out of work. Anyone and their grandmother can write pseudocode. If great aunt Ellen needs a recipe database, she would just have to type "MAKE RECIPE DATABASE" and her computer would have one shortly. Even having programmers around to repair the compiler is unnecessary as we can just simply enter a line "DO MAINTENANCE ON COMPILER" and it will do it.

Finally, we have discovered some startling facts about computer science theory in using our compiler. Not only have we found algorithms that solve SAT in linear time and factor prime numbers in constant time, we also have programs that solve the halting problem as well as answer the question "Is it possible for me to get a date for Saturday night?" These facts have driven most of this research group mad, and with our last vestige of sanity, we are setting fire to the lab. Pray that the Monkey Overlords do not stop us.

Conclusions:

Monkeys like bananas.