Cassowary PureScript Proposal

Contact: Athan Clark
Email: athan.clark@gmail.com
Date: June 3, 2015

Pure Functional Programming for Total Correctness

The benefits of functional languages like PureScript for implementing complex, mathematically dense concepts, like constraint solvers or compilers.

It has been recently discovered that computer science is the heart of mathematics and logic. However, this is true only for functional programming languages like Haskell and PureScript - they do away with the concept of a machine, and direct all attention to the concept of a mathematical function. Programming then becomes the creation of logical specifications and stateless functions, rather than sequentially manipulating data.

Correctness

Through functional programming, you can be garunteed software correctness. The insurance comes from the precise specifications of *types*, detailing the behavior of stateless functions. If an inconsistency is found, the program will simply not compile - it is a safeguard far more effective than unit tests. By establishing *algebraic properties* (such as associativity and commutativivty of expressions), we can **eliminate large classes of bugs** from being possible.

Performance

Functional languages like Haskell or PureScript find an elegant balance between formal correctness and performance. By relying on purity - the lack of affecting reality - one is free to **optimize at a macroscopic level**. This is evident in the premier Haskell compiler, GHC, where you can get better SIMD performance than hand and machine-optimized C. Likewise, Warp (a web server written in Haskell) can achieve over 20,000 responses per second on a $200 laptop, without optimizations enabled, compared to Node.js getting only 600 rps on the same machine.

Clarity

In addition to more correct software and greater performance, purely functional programs are smaller and more concise than their impertative counterparts. With less room for human error and more ability to express ideas, writing many useful algorithms like quicksort and factorial can be done in a single line of code.

Examples

The benefits of using such a language are obvious for any software project, *especially* when problems get complex.

CompCert is a C compiler, similar to GCC, implemented in a functional programming language. It's performance is not as good as GCC's, but it is almost there - about 80%. But the size of the source code is remarkable: CompCert is at about 60,000 lines of code - 50,000 dedicated to exhaustive proofs (verifying correctness of binary translations on all supported architectures), with about 8000 lines actually implementing the compiler. This is hardly a large codebase, especially compared to the 15 million lines of code behind GCC.

Less code *may* mean less room for bugs, but the real reason for functional programming's elegance is its underlying theory - one can directly translate category theory specifications to programs, or formal proofs in propositional logic to types, in an identical syntax. Cryptol is a great example of how someone can define cryptographic algorithms that mirror their algebraic specficiations, one-for-one.

Conclusion

In essence, if you are doing anything complicated, it is best to have a mathematical concept of your problem. And if you have a mathematical definition of a problem, it is **trivial** to implement the algorithm in a functional language, because it is so closely related to mathematics itself. There have been many successful projects implemented in pure functional programming languages - automated trading systems for the stock market, compilers and language interpreters, constraint solvers, and a wide array of mathematically dense algorithms.

Cassowary is a great example of an algorithm that has rich mathematical heritage, but has grown rotten from neglect - it is simply too difficult to take a high-level algebraic concept and implement it in an imperative setting. But with the rise of pure functional programming, I am confident that I can make a smaller, more correct, and more performant implementation of the Cassowary algorithm in PureScript for client-side browsers.

# # #