Seminar Configurable Systems

Prof. Zeller / Christian Lindig — Vesta

Summary

Caching Function Calls Using Precise Dependencies by Heydon, Levin, and Yu presents a caching architecture for an untyped functional programming language. The language is part of a software repository and build system called Vesta, where the language is used to describe the build process. The interpreter implementing the language maintains a cache that it consults before it evaluates a potentially expensive function call. If possible, it uses the cached result from a prior invocation of the function.

The paper solves the problem how to record the dependencies between cache entries, the statically known function body, and only dynamically known values of function parameters. The main contribution of Heydon et al. is a two-level cache architecture which uses cryptographic hashes over values for keys, and syntax-directed rules for calculating dependencies.

A program in the Vesta language specifies how to build a program from source files by calling (external) compiler and linkers in the right order. To update the system after the change of a source file, the entire program conceptually is evaluated again. However, the interpreter now uses cached results such that the amount of work is only proportional to the size of the change rather the size of the system. Therefore, the efficiency of an incremental build in Vesta is delegated to the interpreter's implementation of caching.

Heydon et al. claim a better performance and scalability than Make and support the claim with some experiments. They attribute the improved performance to Vesta's ability to cache the results of high-level functions where Make has to prove bottom-up that files don't need to be updated. While the better performance is evident it is less clear why Make is slower; the paper lacks a detailed study for the reasons.

Because the paper focuses on dependencies and cache architecture it is difficult to compare the Vesta language with other build tools. In particular, concepts like scoping, modules, and the handling of directory structures cannot be judged but only guessed.

The most interesting aspect of the paper is that a build specification is very much like an ordinary (functional) program. Unlike in Make, no notion of dependency exists at the langauge level. Dealing with incremental recompilation in Vesta becomes an optimization problem for the implementation. While the paper does not present a larger example to get a feel for the advantages of this approach, I expect the absence of dependencies at the language level to be a simplification. This simplification for the user comes at the price of an increased complexity of the Vesta implementation. It is fair to say that Vesta is not so much a tool like Make or Cons, but a system. Vesta comes with its own file system which it uses to observe the dependencies of call that invoke external tools such that the user is not forced to specify them.