

Active Libraries

Todd Veldhuizen

April 24, 2004

Active libraries provide both domain-specific abstractions and the know-how needed to optimize them and check their safety requirements. Program generation is an important part of this puzzle: active libraries need to generate customized components to meet the needs of high-performance applications such as scientific computing. The traditional, transformational approach to optimization can often be supplanted by ‘generative optimization,’ in which techniques from program generation and automatic programming are used to synthesize fast implementations from high-level code. The aim of this research program is to produce languages and compilers to support such libraries. Part of this goal is to breathe new life into the old dream of ‘universal languages’ by searching for languages that are universal not just with respect to syntax (i.e., the ability to emulate the surface appearance of any language with, for example, macros), but also universal with respect to staging and safety. Languages satisfying this extended notion of universality ought to be capable of emulating the staging semantics and safety judgments (e.g. type system) of any other language, and would offer a useful base for program generation and active library research.

The technical approach being investigated centers around a new style of optimizing compiler design. Superanalysis, which roughly means combining program analyses by simultaneous coinduction, allows compilers to act as complete decision procedures for approximate axiomatizations of language semantics. Building on this strategy, we have developed a proof technique, Guaranteed Optimization, for designing compilers that compute normal forms of programs. These normal forms can be proven optimal with respect to size, operation counts, and abstraction penalty (but with a restricted notion of program equivalence). Compilers with ‘guaranteed optimization’ can provide language capabilities important for active libraries,

such as staging and generative programming. To achieve flexible safety policies we are investigating embeddings of proof calculi in language semantics, so that proofs are intermingled with code and the optimizer does double-duty as a semi-automated theorem prover. Proof embeddings offer promise as a method for active libraries to enforce domain-specific type systems and safety policies, and for lightweight verification.