The Magnolia Programming Language

Anya Helene Bagge

March 15, 2008

Magnolia is a new language intended for experimentation in programming language design, program transformation and optimisation. It is meant to be general-purpose, but is motivated by the numerical programming domain (in particular, coordinate-free numerics [Bjø00]), where high performance and support for parallel architectures is critical. We have a numerical software library with a number of computationally intensive simulation applications [HFJ99], which is being ported to Magnolia.

Magnolia has much in common with new languages like Fortress [ACH+] and to some degree X10 [CGS+05]. Our focus is different, however, in that we aim to provide a basis for programming language research, rather than a full replacement for existing languages.

Features

C++ based: In the sense that the compiler outputs C++ code (as well as code for some special purpose languages), and that we aim to have a more or less C++ compatible core language, so that each new language feature can be presented as an extension to a simple C++-like language. C++ was chosen because there are good high-performance compilers for it, and because we already have a large body of code in C++.

Signature-based: Operations and types in the language may be manipulated at the signature (interface) level, according to axioms defined on the signature. The implementation is hidden behind the signature, and may look quite different from the signature presented at the point of use. For example, sorting may be implemented as an in-place sort procedure, yet still be used transparently as a function returning a freshly sorted array. A new implementation can be used as a drop-in replacement, as long as it fulfils the signature and satisfies its axioms. This allows us to build a library of replaceable parts with different performance characteristics. Optimisation rules may be used to try and choose the best implementation for a particular situation.

Procedures and functions: A clear distinction is made between procedures (which may change their arguments) and functions (which are 'pure', and may not change their arguments). Through a process of *functionalisation* we can derive function signatures from procedure signatures (where updated arguments are mapped to return values) – thus making procedures available as functions to the programmer. The reverse process of *mutification* transforms expressions written with function calls into a series of procedure calls, possibly introducing temporary copies of data. Certain operations – like IO – involving uncopyable objects may not be subjected to functionalisation/mutification. Programmers

are encouraged to write their code as function calls as much as possible, since analysis and optimisation is easier to perform on pure functional expressions.

Partiality and alerts: Alerts [BDHK06] provide a unified interface to failure handling mechanisms like error return codes, global flags and exceptions. Alerts may also be used to add invariants to procedures and functions. Alert handling policies may be introduced either locally in an expression or procedure, or at the module or global level. Policies can specify substitution of default values, running of special handling code (e.g., to roll back partially completed operations), or propagation of the alert.

Axioms and rewrite rules: Axioms [BH08] are statements about the operations in a program that should be true – often used in program verification. In Magnolia, axioms can be used to derive rewrite rules to use for optimisation. For example, a matrix library may come with axioms that are used as algebraic simplification rules for matrices. Axioms are also used for systematic testing of programs. Furthermore, we are exploring the use of axioms to determine that components are interchangeable.

Optimisation: The user base for Magnolia will be programmers in high-performance computing, so optimisation is critical. The language has no assumption on evaluation order, and the compiler is free to aggressively eliminate calls it determines to be unnecessary. Normally calls that have side-effects – like printing results – must be preserved, but the compiler is free to eliminate calls with 'insignificant' side-effects, like debug printing. The language encourages programmers to explicitly state axioms for user-defined data types so that algebraic simplification may be applied to them; and constructs like data-dependency based loops enables fast computation and automatic parallelisation.

The Project

Magnolia development is just starting, though many features have been explored previously as extensions of C++. The high development costs of C++ extensions, and the difficulty of integrating them nicely with the language is the reason we have decided to work on a new language. Our goal is to experiment with new and interesting language ideas – in particular features that aid programmer productivity and code reusability – and trying them out on non-trivial, real-world applications in order to determine their usefulness.

My role: I am the principal designer of the language, including the not-so-exciting language core, and some of the experimental features. I have done much, if not most of the design work on the above mentioned features (as far as they differ from previous language designs). I am also doing much of the implementation work on the compiler. The language is developed in a group, with other members working on specific features or implementation parts, or being (potential) users of the language. Design ideas are discussed in the group.

References

[ACH+] Eric Allen, David Chase, Joe Hallett, Victor Luchangco, Jan-Willem Maessen, Sukyoung Ryu, Guy L. Steele Jr., and Sam Tobin-Hochstadt. *The Fortress Language Specification*. Sun Microsystems, Inc.

[BDHK06] Anya Helene Bagge, Valentin David, Magne Haveraaen, and Karl Trygve Kalleberg. Stayin' alert: Moulding failure and exceptions to your needs. In *Proceedings of the 5th International Conference on Generative Programming and Component Engineering (GPCE'06)*, Portland, Oregon, October 2006. ACM Press.

[BH08] Anya Helene Bagge and Magne Haveraaen. Axiom-based transformations: Optimisation and testing. In *Proceedings of the 8th Workshop on Language Descriptions, Tools and Applications (LDTA'08)*, Budapest, Hungary, April 2008.

[Bjø00] Petter Bjørstad, editor. *Coordinate-Free Numerics*, volume 8 of *Scientific Programming*. 2000.

[CGS+05] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications*, pages 519–538, New York, NY, USA, 2005. ACM.

[HFJ99] Magne Haveraaen, Helmer André Friis, and Tor Arne Johansen. Formal software engineering for computational modelling. *Nordic Journal of Computing*, 6(3):241–270, 1999.