Requirement Oriented Programming
Concepts, Implications, and Algorithms

David Musser¹, Sibylle Schupp¹, and Rüdiger Loos²

¹ Department of Computer Science
Rensselaer Polytechnic Institute
{musser,schupp}@cs.rpi.edu
² Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
loosr@acm.org

Abstract. Generic programming is requirement oriented programming. We study the concept of genericity and generic concepts. Support of generic programming is exemplified by TECTON formalizations of algebraic and STL concepts. We call for a common effort to establish and maintain generic libraries.

Keywords: generic programming, requirement oriented programming, concept description language, specification language, formal verification, standard template library, algebraic concept definitions, TECTON

1 Introduction

We define generic programming as requirement oriented programming. First of all this means that generic programs are minimal with respect to requirements. Secondly, we understand “requirement” in a sufficiently broad sense that, indeed, every important aspect of programs can be formally stated as a requirement.

Traditionally requirements have been limited to specification requirements, including correctness and invariant assertions. Generic specifications describe the minimal assumptions under which the programs are correct, and thus they describe abstract algorithms that may be instantiated in a broad range of applications. Type checking and verification of generic programs pose new challenges but are only one aspect of genericity.

More important and novel are algorithmic requirements. This means that generic algorithms guarantee performance assertions. We address algorithmic requirements in a companion paper [19] on SUCH THAT [18].

Other important aspects of generic programs are related to portability, to allocation of system resources like storage, and to configuration and maintainability. Even documentation can become easier to read if it is done in a generic way, focused systematically on common aspects. We broadly call these requirements software engineering requirements.

Algorithmic and software engineering requirements have to be based on experiments whereas specification requirements are closer to a foundational framework of genericity.

1.1 Organization of the Paper

In the second section we study the concept of genericity in algebra, in algebraic specifications, and in programming languages.

In the third section we give an introduction to TECTON, a language for expressing generic concepts. We give some concept formalizations from algebra and the Standard Template Library, STL [11][12][20].

We have implemented a tool for processing TECTON, which checks concept descriptions for adherence to the languages syntax, sort-correctness, and identifier availability rules, the latter being a significant generalization of scope rules of previous specification (or programming) languages. In the fourth section we report measurements of the current implementation.

The last section puts forward a proposal to establish and maintain generic libraries.

2 The Concept of Genericity

2.1 Related Concepts in Computer Science

The concept of an algorithm as defined by Knuth [11] is characterized by a set of five properties: definiteness, finiteness, input, output, and effectiveness. As observed by Collins [4], relaxing the requirement of definiteness—that each step of an algorithm must be precisely defined—leads to the concept of an abstract algorithm or an algorithm schema, earlier terms for what is today more frequently referred to as a generic algorithm. Thus an algorithm in the traditional sense can be regarded as a generic algorithm for which all sources of indefiniteness—genericity—are made definite.

2.2 Related Concepts in Algebra

In the nineteen twenties, led by Emmy Noether, Grete Herrmann, Emil Artin, and others, algebra turned from working in concrete domains to abstract algebra and abstract algebraic domains. The intention was to base the proofs of the theorems on minimal requirements and to peel away from their inherent generality all additional, non-essential features available in concrete domains. Their student van der Waerden published this fresh view of algebra under the title Modern Algebra, meaning abstract algebra. This book was so successful, with numerous editions appearing over many years, that van der Waerden eventually decided to rename it simply Algebra.

We see a similar turn from the concrete to the abstract in programming. We expect that following the same principles “generic programming” will become so popular that one day it will be called simply “programming.”

We also are convinced that “modern” algebra is the adequate theoretical foundation of genericity.\footnote{We maintain the distinction between the algebra-based notion of genericity and the set theoretical notion of type polymorphism.}

1