# Research on a New Abstract Programming Language for Generic Constraints

Z.K. Zuo, C.J. Wang, H.H. Shi
College of Computer Information and Engineering
Jiangxi Normal University
Jiangxi, China
Provincial Key Laboratory of High Performance Computing Technology
Jiangxi Normal University
Jiangxi, China

W.P. Xie
Provincial Key Laboratory of High Performance Computing Technology
Jiangxi Normal University
Jiangxi, China
State Key Laboratory of Software Engineering,
Wuhan University
Wuhan, China

Z. You
Provincial Key Laboratory of High Performance Computing Technology
Jiangxi Normal University
Jiangxi, China

J.X. Lu
College of Computer Information and Engineering
Jiangxi ,Normal University
Jiangxi, China

*Abstract*—**Generic programming has emerged as a paradigm for the development of highly reusable and safe software libraries. Generic constraint mechanism explicitly describes which concrete types can instantiate generic types. As it can detect and verify the validity of generic parameter instantiated, thereby guarantee dependability and safety of generic programming. Many modern programming languages support basic generic constraints. The paper reports on a comparison of facilities for generic constraints in five programming languages: C++, ConceptC++, Java, C#, named type constraints, and then presents a new abstract programming language: Apla, which supports for generic constraints base on algebraic structures. As language increasingly support generic constraints, it is necessary that language designers understand the constraint features to enable the effective use of generics and that their absence can cause difficulties for programmers.**

***Keywords-abstract programming language; generic constraints; Apla; algebraic structures***

## I. Introduction

Pioneered by Alexander Stepanov and David Musser, generic programming[1, 2] is a programming paradigm for developing efficient, reusable software libraries. Generic constraints mechanism includes a collection of features for constraining generic parameters and verification of the validity of generic parameter instantiated, thereby guarantees dependability and safety of generic programs. Actually, we believe that generic constraints are the key points to solve the reliability of GP(Generic Programming). Many modern programming languages all add mechanisms of generic constraints, such as C#, java and Concept C++ [3], etc. However, the constraints of these languages are regarded as narrow constraints at most [4]. They do support for syntax requirements, but not for semantic requirements [5]. This paper presents generic constraints of a new abstract programming language, which is called Apla(Abstract Programming LAnguage), and then displays the implementation of automatically transforming the abstract language to C++ codes based on C++ generator.

The organization of this paper is as follows. In section 2, we illustrate the related works of generic constraints. The section 3 presents a new description of generic constraints of Apla language by means of algebraic structures, and then describes one typical case using the Apla constraints mechanism in section 4. Finally, we draw conclusions and figure out future work in section 5.

## II. Related Works

Musser [6] presented generally recognized definition of generic constraint.

**Definition 1.** Generic constraint

Given A is a set of abstractions (such as Abstract Data Type (ADT)), and generic constraint C is a set of requirements R. That is:

$$C(R/A)=\{a\in A\mid \forall r\in R, r(a)=true\}.$$

In which, A is a set of all types in the domain, and R is commonness of all elements in C.

C++ templates are the core of the design of current successful mainstream libraries and systems [3]. However, they lack formal descriptions of generic constraint. Compare to C++ templates, concepts of ConceptC++ generate sets of primitive operations with signatures from a simpler and more abstract language and adds formal description of generic constraints explicitly. C#/Java generics can provide enhanced safety but are also somewhat limited in capabilities [7]. They introduce F-Bounded Parametric Polymorphism to constrain type parameters. Type variables in a parameterized type can be bounded by a class or an interface, and then only sub-types of the bound can be used to instantiate the parameterized type.

For instance, there is an example of C# generics, as shown in example 1.

**Example 1.**

*public class LinkedList<K, T> where K: IComparable <K>*
*{   T Find (K key);    }*

Parameterized type K is supposed to implement interface IComparable. It means that any specific type which does not implement interface IComparable cannot instantiate the parameterized type K.

There is another example of Java generics, as shown in example 2.

**Example 2.**

*public static <T extends Object & Comparable<T>> void sort (T [] array)*

Java generics are similar to C# generics. It means that only sub-types of Object and Comparable<T> can be used to instantiate the parameterized type T.

Swen Bing proposed named type constraint mechanism [4]; selected C++ language as the host language; designed the standard constraints library; developed a compiler front-end (has been integrated into the C++ front), as far as possible to reuse existing C++ resource [4].

## III. APLA GENERIC CONSTRAINT MECHANISM

Based on PAR(Partition And Recur) method [8], this section puts forward Apla [9, 10, 11] generic constraint mechanism. With abstract programming language Apla for the host language, the mechanism focuses on algebraic structure semantic constraint. Compared with the existing relevant work [3, 4], the approach presented in this section improves the abstract degree of generic constraint, and supports semantic constraint. Thus, it facilitates formal verifying.

Abstract data type defines a collection of data and a set of operations on the data set. It can be viewed as corresponding to a series of elements and their algebraic operations. Operational semantics can be described by using algebraic equation axioms. As the different operations and equation axiom can form all kinds of algebraic system specification, any abstract data type can be described by algebraic system specification. We proposed to construct the appropriate algebraic structure, which unifies a class of problems in a mathematical model or a library of reusable programming components.

We add generic constraints mechanisms into Apla language. It includes three parts: constraints_definition, constraints_call and constraints_instantiation. Based on constraints_definition, we design a predefined Apla algebraic structure generic constraint library. It defines idempotent structure, commutative structure, reversible structure, groupoid, semigroup, monoid, Abelian monoid, group, Abelian group, ring, semiring, and a series of other constraints. Based on the predefined basic algebraic structure constraint library, users can define more algebraic structure constraints according to the custom demand. Then users can constrain that parameterized types and functions should conform to constraints requirements in the step of constraints_call. At last, constraints_instantiation is instantiating generic types into specific types based on constraints.

There is BNF description of Apla generic constraints_definition:

*constraints_definition ::=*
*"define constraint"*
*<constraints_name>(<constraints_declaration>);*
*[<constraints_refinement>]{definition_body}*
*"enddef"*
*constraints_declaration::=*
*<generic_ADT>;*
*"generic" <"someADT" <type_parameter>*
*{;"someADT" <type_parameter>}*
*constraints_refinement::=*
*"where"(<constraints_call>)*

There is BNF description of Apla generic constraints_call:

*constraints_call::=*
*"where" <constraints_name>*
*(<ADT_name>(<data_type_parameter>,<generic_operation>))*
*{<logical_operator><constraints_name>*
*(<ADT_name>(<data_type_parameter>,<generic_operation>))}*

There is BNF description of Apla generic constraints_instantiation:

*constraints_instantiation::=*

*"procedure"<instantiation_procedure_name>*

*:"new"<generic_procedure_name>*

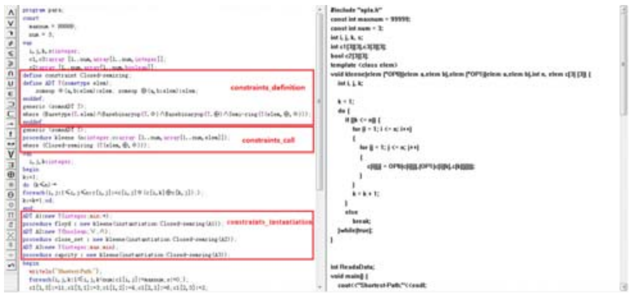
*|"function"<instantiation_function_name>*

*:"new"<generic_function_name>*

*{"instantiation" <constraints_name>[{<instantiation_type>;}]}*

## IV. CASE STUDIES

There is one case of closed semi-ring. It is called kleene algorithm [12], which unifies a family of path problems, including all-pairs shortest path, transitive closure, and maximum capacity path, defined on directed or undirected graphs [13].It is shown in figure 1.



```
program path;
const
  maxnum = 99999;
  num = 3;
var
  [illegible]
define constraint Closed-semiring;
define ADT T(sometype elem);
  [illegible]
enddef;
generic (someADT T);
where [illegible]
enddef;
generic (someADT T);
procedure kleene [illegible]
where (Closed-semiring (T(elem, ⊕, ⊗)));
var
  [illegible]
begin
[illegible]
end;
[illegible]
procedure floyd : new kleene(instantiation Closed-semiring(A1));
[illegible]
procedure close_set : new kleene(instantiation Closed-semiring(A2));
[illegible]
procedure capacity : new kleene(instantiation Closed-semiring(A3));
begin
  writeln("ShortestPath");
  [illegible]
```

```
#include "apla.h"
const int maxnum = 99999;
const int num = 3;
int i, j, k, n;
[illegible]
[illegible]
template <class elem>
void kleene(elem [illegible]
  int i, j, k;

  k = 1;
  do {
    if ([illegible]) {
      for (i = 1; i <= n; i++)
      {
        for (j = 1; j <= n; j++)
        {
          [illegible]
        }
      }
      k = k + 1;
    }
    else
      break;
  }while(true);
}

int ReadaData;
void main() {
  cout<<"Shortest-Path"<<endl;
```

FIGURE I. AUTOMATICALLY GENERATING KLEENE C++ TEMPLATE PROGRAM

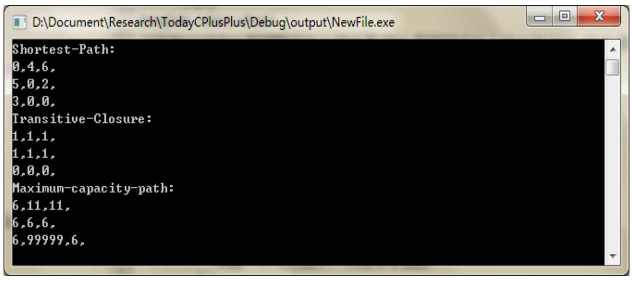
(1)constraints_definition.

Basetype, Basebinaryop and Semi-ring belong to Apla generic constraint library, which is predefined. The constraints_definition of closed-semiring directly call them.

(2)constraints_call.

(3)constraints_instantiation.

We can generate the shortest path algorithm, the transitive closure algorithm and the maximum capacity algorithm respectively.


D:\Document\Research\TodayCPlusPlus\Debug\output\NewFile.exe

```
Shortest-Path:
0,4,6,
5,0,2,
3,0,0,
Transitive-Closure:
1,1,1,
1,1,1,
0,0,0,
Maximum-capacity-path:
6,11,11,
6,6,6,
6,99999,6,
```

FIGURE II. PROGRAM EXECUTION RESULT

Figure 1 shows us the result generated by the C++ generator. The left part of the figure 1 is Apla generic kleene program. It includes constraints_definition, constraints_call and constraints_instantiation. The right part of the figure 1 is the part of C++ codes that is generated by the C++ generator automatically. Figure 2 presents execution of the C++ codes.

## V. CONCLUSION AND FUTURE RESEARCH

Generic constraints mechanism includes a collection of features for constraining generic parameters and verification of the validity of generic parameter instantiated, thereby guarantees dependability and safety of generic programs. The paper describes the current research situation of generic constraints. It is difficult to describe and verify generic programs with dynamic semantic constraints. We present generic semantic constraints of a new abstract programming language based on algebraic structures, then one typical case is given. Finally, we transform the case to C++ template program automatically by C++ generator. It has been found that the generic constraints mechanism can solve a series of complex generic constraints problems, so markedly improves dependability and safety of generic programs.

As to the constraint verification of Apla generic program, we will write another paper to demonstrate.

## ACKNOWLEDGEMENTS

This work was financially supported by the National Nature Science Foundation of China (61462039, 61363012, 61363013, 61462041 & 61020106009) and Natural Science Foundation of Jiangxi Province of China (20142BAB217023, 20142BAB217026, 20142BAB207026 & 20132BAB211022).

## REFERENCES

[1] Garcia, R., Jarvi, J., Lumsdaine, A., Siek, J. & Willcock, J., An extended comparative study of language support for generic programming. *Journal of Functional Programming*, **17(2)**, pp. 145-205, 2007.

[2] Zuo, Z.K., Xue, J.Y. & Wang, C.J., Closed semi-ring constraint verification of generic Kleene algorithm. *Journal of Computational Information Systems*, **9(22)**, pp. 9047-9054, 2013.

[3] Gabriel, D.R. & Bjarne, S., Specifying C++ concepts, *Proc. of Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, New York: ACM Press, pp. 295-308, 2006.

[4] Swen, B., Object orientation, generic programming and type constraint checking. *Chinese Journal of Computers*, **27(11)**, pp. 1492-1504, 2004.

[5] Chen, L. & Xu, B.W., Using static analysis to extract C++0X concepts. *Chinese Journal of Computers*, **32(9)**, pp. 1792-1803, 2009.

[6] Musser, D.R. & Stepanov, A.A., Generic programming, *Proc. of Symbolic and Algebraic Computation*, LNCS 358, Berlin, Heidelberg: Springer-Verlag, pp. 13-25, 1989.

[7] Zuo, Z.K., Xue, J.Y. & Wang, C.J., Constraint verification of generic algorithmic program for solving general network path problems. *Journal of Networks*, **8(5)**, pp. 1050-1057, 2013.

[8] Xue, J.Y., PAR method and its supporting platform, *Proc. of the 1st International Workshop on Asian Working Conference on Verified Software*, Technical Report, 348, Macao:UNU-IIST, 2006.

[9] Wang, C.J., Luo, H.M. & Zuo, Z.K., Formal software specification generation approach based on problem patterns. *Journal of Computer Research and Development*, **50(2)**, pp. 352-360, 2013.

[10] Shi, H.H. & Xue, J.Y., Research on automated sorting algorithms generation based on PAR. *Journal of Software* , **23(9)**, pp. 2248-2260, 2012.

[11] Zuo, Z.K., You, Z. & Xue, J.Y., Derivation and formal proof of non-recursive postorder binary-tree traversal algorithm. *Computer Engineering and Science*, **32(3)**, pp. 119-123, 2010.

[12] Li, Y., Yu, S.C. & Wang, P., Research and realization of generic identity based on functional language. *Computer Engineering and Applications*, **48(28)**, pp. 71-76, 2012.

[13] Chen, L., Xu, B.W.,Qian, J., Zhou, T.L. & Zhou, Y.M., Refactoring generic instantiations based on type propagation analysis. *Journal of Software*, **20(10)**, pp. 2617-2627, 2009.