International Journal of Scientific and Innovative Research 2014; 2(1): 67-70,
P-ISSN 2347-2189, E- ISSN 2347-4971

IJSIR

PROBLEM ANALYSIS DIAGRAM DECLARATIONS OF COMPILER TECHNIQUE FOR APPLICATIONS OF C/C++ PROGRAMMING

ROHIT SAXENA[1], DEEPAK SINGH[1], *AMOD TIWARI[2]
[1]Rama Institute of Engineering and Technology, [2]Bhabha Institute of Technology, Kanpur Dehat, Kanpur, Uttar Pradesh ,India

*Address for correspondence: Dr. Amod Tiwari, Director- Professor , Bhabha Institute of Technology, Kanpur Dehat, Kanpur, Uttar Pradesh ,India,
e mail : amodtiwari@gmail.com

ABSTRACT

Language transformation is challenged part of programming into runtime verification tools. To increase the idea during the writing concept that these runtime verification tools can be used for testing realworld programs, the paper uses compiler technique, a subset of the C and C++ programming language, which can be used to execute and test real programs. Compiler technique is extended with threads and synchronization construction, and two concurrent semantics are derived from its sequential semantics. First one is defining a sequentially consistent memory model can be easily transformed into a runtime verification tool for checking datarace and deadlock freeness. Second one is relatively minimal fashion a relaxed memory model. The paper increased the efficiency of the programming language like C and C++ for using above Programming Analysis Design (PAD) technique.

Keywords: Pseudo-code, Structured programming chart, Top down analysis, Compiler tool.

INTRODUCTION

Problem analysis design (PAD) process of using formal definitions of programming languages as testing and analysis tools. We argue here that variable K [7, 8] definitions can be used to test and analyze executions of programs written concept in real-life languages either directly or by extending them to become runtime analysis tools.The rewriting logic representation of K definitions gives them access to the arsenal of generic tools for rewriting logic available through the Maude rewrite engine [9] state space exploration. This collection of analysis tools is by itself enough to provide more information about the behaviors of a program than one would get by simply testing the program using an interpreter or a compiler for that language. Nevertheless, the effort of defining the semantics pays back in more than just one way: by relatively few alterations to the definitions, one can use the same generic tools to obtain type checkers and type inferencers [9], static policy checking tools [10,11], runtime verification tools [13], and even Hoare-like program verification tools.[12].

It is well known that several charts like NS chart [1], Jackson's chart [2], and Problem Analysis Diagram (PAD) [3] are much more effective to teach structured programming than a classical flow chart, since each stepwise refinement process by top down should be either concatenation, selection or repetition. Especially PAD is suitable to describe complicated programs directly. However, these charts do not contain declaration. It is important in object oriented programming to see how classes are declared. Then we propose a modified PAD including declaration.

INSERTING DECLARATION

In order to introduce PAD, consider the following C program containing pseudo-codes which finds the maximum and the second maximum number from ten input numbers.

```
//Program 1
#include <stdio.h>
        int main(){
        int x[10],x1,x2;
        <* input x *>
        <* (x1, x2) from (x[0], x[1]) *>
        for(i=2; i<10; i++)
                <* (x1, x2) from (x1, x2, x[i]) *>
        <* output x1, x2 *> return 0;
}
<* (x1, x2) from (x[0], x[1]) *>:={
        if(x[0]<x[1]){x1=x[1]; x2=x[0];}
        else {x1=x[0]; x2=x[1];}
}
<* (x1, x2) from (x1, x2, x[i]) *>:={
        if(x2<x[i]){x2=x1; x1=x[i];}
        else if(x1<x[i]){x1=x[0];}
}
```

The corresponding PAD is shown in Figure 1, where B1= {x1=x[1]; x2=x[0];}, etc.

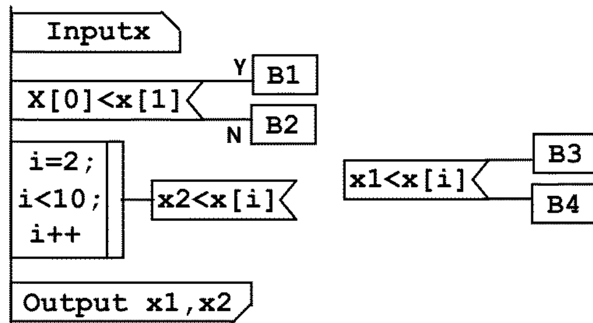

Figure 1: Problem analysis diagram of Program 1

In Figure 1, declarations are not contained in PAD. In order to discuss how to include declaration in a structured programming chart, consider the following program.

```
//Program 2
#include <stdio.h>
int sum(int u, int v){return u+v;}
void add(int* p, int* q){*p += *q;}
    int main(){
            int x,y; <* input x,y *>
            add(x, y);
            <* output x,sum(&x,&y) *> return 0;
}
```

There are some design policies of the compiler program using with application of C and C++, All of the necessary information is included, Declaration blocks are not similar to execution blocks, Pseudo-codes are explicitly distinguished and Easy to draw with popular application software, we propose a chart as shown in Figure 3.

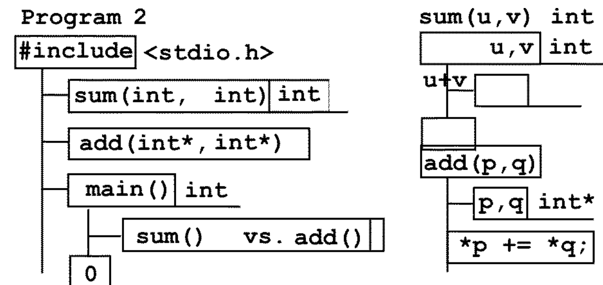

Figure 2: Declaration and definition of Program

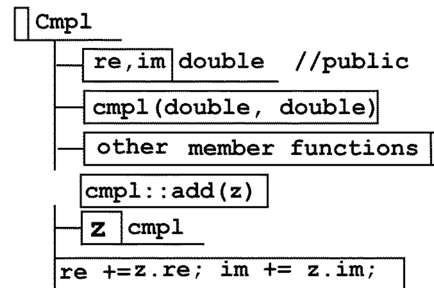

Figure 3: A class and its member function

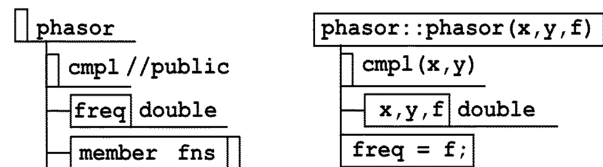

Figure 4: A derived class and its constructor

The chart of derived classes is shown in Figure 5.A sample of member functions is phasor::sequence (FILE*) which outputs the time-sequence of the sinusoidal wave.

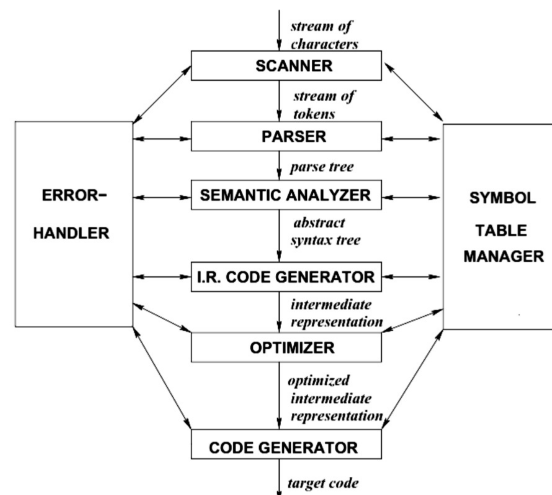

Figure 5 shows a declaration of a class cmpl and the definition of its member function add (cmpl*).

EFFECT OF MODIFICATION

The advantage to use a chart compared with a colored text is reduction of reserved words as shown in fig. 5, which implies large characters can be used in a lecture using MS Power Point slides.

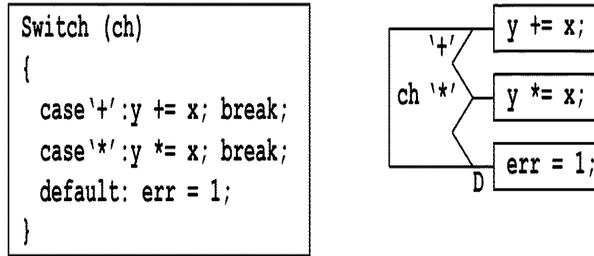

Figure 6: Effect of eliminating reserved words

Although PAD is a compact expression of an algorithm as shown in Figures 1 and 5, alternative expression in Figure 6 is more compact and easy to draw, since every chart can be drawn by superposing polylines on a uniformly spaced plain text.

Remark <* input x *> in Program 1 represents each x[i](0 <= i < 10) should be filled by an integer using a certain method. An example of specified input/output is shown in Figure 7 which correspond to <* input (fscn): x *> and <* printf ("%8d\n"): x1, x2 *>.

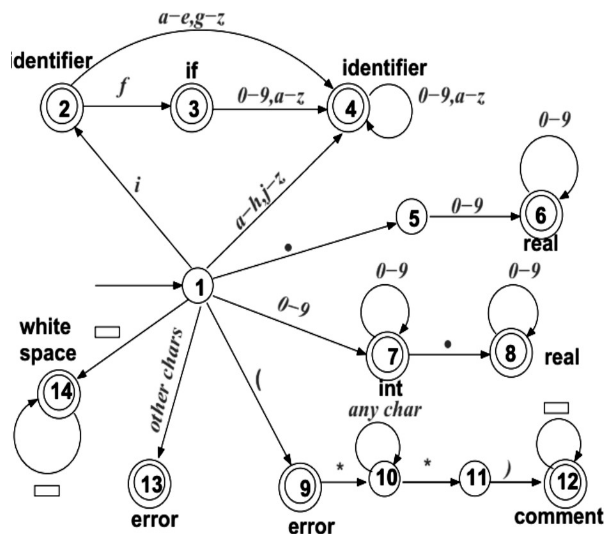

Figure 7: Alternatives for compound statements chart with error

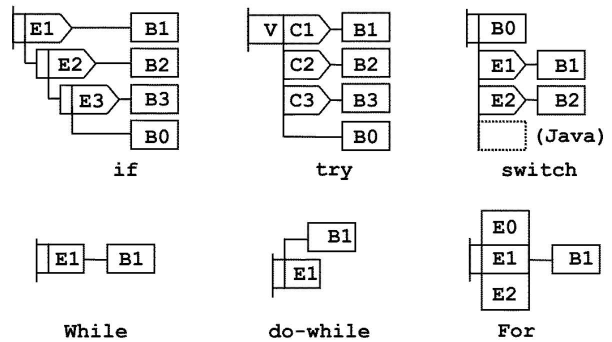

Figure 8: Alternatives for compound statements code

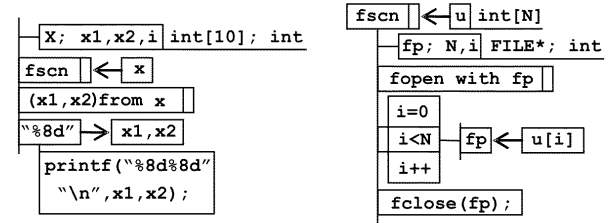

Figure 9: An example of input and output for Program

CONCLUSION

We have shown how K definitions of programming languages can be turned (with negligible effort) into runtime analysis tools for testing and analyzing executions of concurrent programs. We do not claim here that the tools one obtains almost for free within the K framework completely eliminate the need of writing dedicated analysis tools in "real" programming languages. The proposed chart is an object oriented programming chart rather than a structured programming chart i.e. (C ,C++). Since reserved words are reduced in such a chart, it is easier than a colored text to see how the algorithm is implemented.

REFERENCES

1. Weiss, E. H., Visualizing a Procedure with Nassi-Schneiderman Charts. Journal of Technical Writing and Communication 1990 ; 20(3) : 237-254.
2. http://www.cbu.edu/~lschmitt/I351/Nassi%20Schneiderman.htm (Mark Kelly, Structured Design with Nassi-Schneiderman Charts)
3. Jackson, M. A., Principle of Program Design, Academic Press, 1975.
4. http://www.informingscience.org/proceedings/IS2003Proceedings/docs/091Ourus.pdf (N. Ourusoff, Using Jackson Structured Programming (JSP) and Jackson Workbench to Teach Program Design)
5. Futamuta, Y., Kawai, T., Tsutsumi, M., and Horikoshi,

H., Development of computer programs by Problem Analysis Diagram (PAD). In the Proc. 5th Int'l Conf. on Software Engineering, New York, IEEE Computer Soc., 1981, pp.325-332.

6. http://fi.ftmr.info/PapersToRead/PAD-JARECT.PDF (Y. Futamura and T. Kawai, Problem Analysis diagram)

7. Rosu G. K Overview and SIMPLE Case Study. Electronic Notes in Theoretical Computer Science. 2014; 304: 3–56.

8. Rosu, G. and T. F. Serbanuta. An overview of the K semantic framework. Journal of Logic and Algebraic Programming 2010 ; 79: 397–434.

9. Clavel, M., F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet and C. Talcott. All About Maude, A High-Performance Logical Framework, LNCS 4350, Springer, 2007.

10. Hills, M., F. Chen and G. Rosu , A rewriting logic approach to static checking of units of measurement in C, in: RULE'08, 2008, pp. 76–91, Tech. Rep. IAI-TR-08-02, Institut für Informatik III, Rheinische Friedrich-Wilhelm-Universität Bonn.

11. Hills, M. and G. Rosu, A rewriting logic semantics approach to modular program analysis, in: RTA'10, LIPIcs 6 (2010), pp. 151–160.

12. Rosu, G., C. Ellison and W. Schulte, Matching logic: An alternative to Hoare/Floyd logic, in: AMAST '10, LNCS 6486, 2010, pp. 142–162.

13. Rosu, Runtime veriûcation of C memory safety, in: RV'09, LNCS 5779, 2009, pp. 132–152.