Verifying the Correctness of HARPO Programs

Inaam Ahmed
Electrical and Computer Engineering
Memorial University of Newfoundland
Email: inaama@mun.ca

Theodore S. Norvell
Electrical and Computer Engineering
Memorial University of Newfoundland
Email: theo@mun.ca

R. Venkatesan
Electrical and Computer Engineering
Memorial University of Newfoundland
Email: venky@mun.ca

***Abstract*- This paper reports an implementation of a verifier for HARPO, a concurrent programming language. The verifier translates annotated programs into the intermediate verification language Boogie; the translated program is then checked by the Boogie verifier. The translation of HARPO programs is achieved by parsing the source code into valid abstract syntax tree; each node of the syntax tree is then used for generating its equivalent Boogie code. By using our verifier, HARPO programmers can be assured that their designs are free of defects.**

Index Terms— HARPO, Boogie, Program Verification

I. INTRODUCTION

Defects in software cost money and sometimes lives. How can we ensure that a program is correct? Testing does not guarantee that a program is correct for every input. Testing concurrent programs does not ensure that a program is correct even for the example inputs. As testing of a program with any sets of inputs can show the presence of errors, but not their absence, testing is insufficient to ensure the system is error free [1]. The purpose of program verification is to provide an error-free software system. A critical system needs to be known to be defect free with mathematical certainty prior to being deployed. The formal program verification considers computer programs as mathematical objects and their properties are verified by mathematical proofs.

HARPO is an object-oriented, concurrent programming language [2,3,4]. HARPO is intended to target a variety of hardware platforms including microprocessors, field-programmable gate arrays, and coarse-grained reconfigurable architectures. The design of HARPO compiler is shown in Figure 0. The HARPO source programs may be compiled to the equivalent C, Boogie, or VHDL.

In this paper, we report the initial design of the verifier implementation which is able to compile HARPO programs into equivalent Boogie programs. The HARPO verifier works by translating HARPO programs to code in the Boogie intermediate verification language [5,7]. The Boogie verification uses a theorem prover, Satisfiability Modulo Theories (SMT) solver Z3 to detect the errors in source code. This verification methodology for program verification based on theorem provers guarantees the accuracy of programs *100%* [6].

Numerous verification tools have been designed to verify sequential and concurrent programs in past few years based on Boogie, including Dafny, Chalice, VCC, Eiffel, and Spec# [5,10].

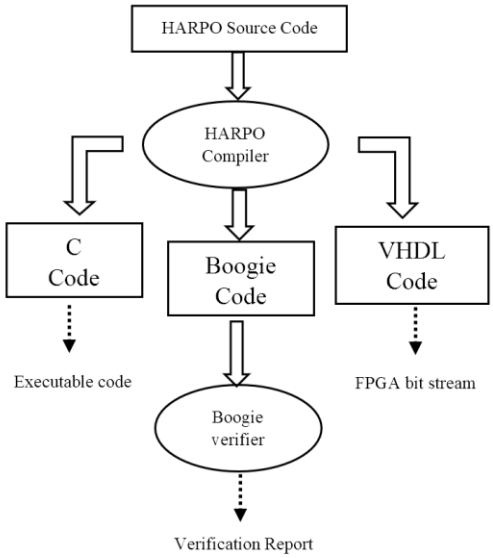

Figure 0: HARPO Compiler translates the HARPO source program into equivalent C language, Boogie[1] Language, and, VHDL. Dotted arrows indicate the final outputs.

HARPO verifier is built top of Boogie which act as an intermediary language to communicate with theorem provers. We have successfully verified a few examples using technique presented in this paper.

This paper is organized as follows: section II describes the annotation for writing HARPO program specification. Section III provides an overview of the translation. Section IV provides an example of translation and finally section V provides conclusion and some future work.

II. HARPO PROGRAM ANNOTATIONS FOR VERIFICATION

HARPO programs consist of components namely, interfaces, classes, objects and constants [3,4]. We introduce ghost fields in HARPO for verification. Ghost fields are ignored by other backends such as C and VHDL [7,8].

Annotations for verification are a part of the syntax of HARPO programs. These annotations are necessary for writing the specifications of the HARPO programs. However, use of annotations must be semantically correct for getting the error report from the Boogie.

[1] Boogie is an intermediate language used for verification named as Boogie IVL, and the Boogie verifier is a tool which takes the programs written in Boogie IVL to verify [5].

Classes and Interfaces: Classes are basic constructs of HARPO as it is an object-oriented language. In HARPO each class may have *claim* and *invariant* annotations. *Claim* annotations on classes indicate which locations are owned by objects of the classes on instantiation. Class invariants are propositions that must be true when the object is not occupied by any thread.

Perm Type: To express ownership of locations *Perm*, a permission type, has been introduced in HARPO it is used for verification only. These *ghost* permission type variables are assigned fractional values between 0 and 1 representing the degree of ownership on fields (locations). The higher the value, the greater the ownership of the location [9]. Permissions on locations are held by objects or threads. A tread must hold a nonzero amount of permission on an object to read it and a permission of *1.0* to write to it. Since the total permission on each location is *1.0*, data races are avoided.

Class Members: A methods specification clause consists of a *pre-condition*, a *post condition;* a *takes PermMap*, a *gives PermMap*, and a *borrows PermMap*. The conditions are Boolean expressions that may contain *ghost* variables. A *post-condition* expression contains variables with apostrophe e.g. if *x* is the initial value then *x'* is the final value.

Threads: Threads are executable blocks of code and each block contains sequence of HARPO commands and local declarations. For verification purpose, a thread may contains *claim* specification. The claim indicates the amount of permission held initially by the thread.

Assertions and assumptions: HARPO program may contains *assume conditions*, and *assert conditions* where *condition* is the Boolean expression. '*assert'* and '*assume'* are useful to write the specification of the HARPO program. They can easily let the verifier skip or put assertion on interested parts of the code for verification.

Loops and Concurrent Blocks: Loops are much like loop structures in other higher level programming languages. Loops need *invariant condition(s)* specification, where condition is Boolean expression. *Parallel blocks* are implemented with *co* keyword. Initially *co* block has claim specification that takes permission values of locations from the *thread* and split it into the inner concurrent blocks. The sum of permissions split by the *co* should not exceed the initial permission held by the parent *thread*.

III. PROGRAM TRANSLATION INTO BOOGIE

A standard approach for program verification is to use the theorem proving technique [5]. Source code with program specifications in a higher programming language is converted into verification conditions. However, generating the verification conditions for theorem provers is a complex task. A common approach to deal with this complexity is to use an intermediate verification language. We have mitigated the complexity by dividing the verification process into two main steps: Translating the program specifications into an intermediate verification language (IVL), Boogie. Later, the Boogie source is converted into verification conditions and checked by Boogie verifier to generate the error report. In our case, Boogie verifier reports back the errors in the specifications of an input Boogie program translated from HARPO program.

HARPO is concurrent, Boogie is sequential and so there is a need to represent the actions of other threads in the generated Boogie code. We do this by havocing locations that may have been changed by other threads at appropriate points in the code for a thread. Finally, we can determine the correctness of concurrent HARPO programs.

A. BOOGIE PRELUDE

The Boogie prelude is part of the Boogie program independent of the source program being translated into Boogie. This part contains some important properties such as modeling memory, reference types, type axioms, array length and permission type, required for translation of HARPO program. The final output consists of Boogie prelude and the translation of specific HARPO programs.

B. MEMORY MODEL

Memory model is an important decision while translating HARPO programs into Boogie. We are using heap memory model which maps the fields and object references to values. For array types we declare a separate heap.

C. TRANSLATION OF PROGRAM COMPONENTS

All the reference types in HARPO are translated into the Boogie types. . A few translations of HARPO program components to Boogie, shown in Table 0. Following are some HARPO program components and their translated definitions in Boogie:

Class: HARPO classes translated into constants in Boogie. In case of a class, implementing an *interface* it is expressed by subtype in Boogie.

Interface: Interfaces are translated into constants in Boogie.

Fields: Fields are translated into unique constant in Boogie. Field name is appended to the class name after '.' character.

Constants: Constants of HARPO translated into constants in Boogie with an axiom referring the well-definedness of expression and equality between value and constant.

Types: HARPO has four different categories of types including, primitive types, reference object types, permission type, and array types. Types are converted in Boogie such as, primitive types converted into Boogie primitive types; reference types into *Ref* types; permission types to *Perm* type*;* and array types to *ArrayRef* type in Boogie.

Expressions: Expressions form with arithmetic and logical operators. All expressions checked with their well-defindness before translating them to equivalent Boogie expressions. Assertions are generated to check that expressions are well defined, for example, that array indices are in bounds.

TABLE 0

HARPO PROGRAM COMPONENTS AND THEIR EQUIVALENT BOOGIE CODE

Program components	***HARPO Code***	***Boogie Code***
Class	***(class A*** *class members* ***class)***	***const unique*** *A: ClassName;*
Interface	***(interface*** *B Interface members* ***interface)***	***const unique*** *B: ClassName;*
Field	***obj*** *h: Int8:* ***=*** *Exp*	***const unique*** *A.h: Field int;*
Constants	***const*** *c: real16: = Exp_h*	***const*** *c: real;* ***axiom*** *x == Exp_b*
While Statement	***(while*** *G_b* ***invariant*** *I statement(s)* ***while)***	***while (***G_b***) invariant*** *I Boogie statements*
Thread	***(thread*** *T* ***claim*** *init_Permission block* ***thread)***	***Procedure*** *A.T(this: Ref)* ***Modifies*** *H. ArrayH;* ***Requires*** *dtype(this) <: C; {…thread block …claim translation}*

Statements: Statements include assignments, *if*, *while*, *co*, *for* and a few more [3]. Prior to the translation of HARPO class, a prefix *this* added to all fields and methods of the class. Local variables of the class are always promoted to the fields in Boogie.

Thread: Thread is a block of code inside a HARPO class annotated with *claim* specification. Multiple threads exploit the concurrency among the objects using *Rendezvous*. For instance, threads are translated into procedure of Boogie containing *this* parameter denoting the object containing the thread.

Methods: Methods have declaration containing the contracts (permission specifications and assertions) and its implementation inside the thread. Implementation of method consists of sequence of HARPO statements. Permission specifications are translated into the permission maps in Boogie by implementing 2D array Heap. Whereas, assertions, *pre condition* and *post condition*, are converted into *requires condition* and *ensures condition* clauses respectively, in Boogie. A *condition* is always boolean expression regardless of it's use in HARPO or Boogie.

IV. HARPO MATH CLASS TRANSLATION

Given a HARPO program in Listing 0 consists of a class having an object declaration *'c'* and a thread **t0**. **t0* is assigning* an expression to *'c'* and making an assertion on the value of *'c'* with an integer *'4'*.

```
(class Math
    obj c:int32 :=0;
    (thread (*t0*)
           c:=2+2;
           assert c=4;
    thread)
class)
```

Listing 0: HARPO program consists of 'Math' class containing object *c* and thread **t0**

```
//prelude
1.  type Ref;
2.  type Field a;
3.  type HeapType = <a> [Ref,Field a]a;
4.  var Heap:HeapType;
5.  type Perm = real ;
6.  type PermissionType = <a>[Ref, Field a]Perm;
// Specific translated part of Listing 0
7.  type className;
8.  function dtype(Ref) returns (className);
9.  const unique Math:className;
10. const unique Math.c : Field int;
11. procedure Math.t0(this:Ref)
12. modifies Heap;
13. { var Permission : PermissionType where
          (forall <a> r:Ref, f : Field a ::
          Permission[r,f] == 0.0  ) ;
14. var oldHeap, tmpHeap : HeapType ;
15. assert Permission[ this, Math.c ] == 1.0 ;
16. Heap[this,Math.c]:= 2+2;
17. assert Permission[ this, Math.c ] > 0.0 ;
18. assert Heap[this,Math.c]==4 ;}
```

Listing 1: Boogie code translation of Listing 0

We have implemented an automation of translation from Listing 0 to Listing 1. The process of automation is explained in Section V. The following semantic errors occur in listing 0.

- The assignment *"c :=2+2;"* has an error because the thread does not have permission of 1.0 on the location.
- Second the assertion *"assert c=4 ;"* has an error. because the thread does not have permission > *0* on the location '*c'*.

Boogie will report the errors in Listing 1 because assertions in lines *15* and *17* will not hold. To remove errors from the program we need to add '*claim c'* specification in '**t0*'*, show in Listing 2. The result of translating Listing 2 is shown in listing 3. The Boogie verifier finds no errors in this code.

V. AUTOMATING THE TRANSLATION

The compiler's parser generates an Abstract Syntax Tree (AST) for the given HARPO program. Figure 1 has

```
(class Math
    obj c:int32 :=0;
    (thread (*t0*) claim c
           c:=2+2;
           assert c=4;
    thread)
class)
```

Listing 2: HARPO program consists of 'Math' class containing object *c* and thread **t0**

```
11. procedure Math.t0(this: Ref)
12. modifies Heap;
  .
  .
  .
15. Permission[this, Math.c] := 1.0;
16. assert Permission[ this, Math.c ] == 1.0 ;
17. Heap[this,Math.c]:= 2+2;
18. assert Permission[ this, Math.c ] > 0.0 ;
19. assert Heap[this,Math.c]==4 ;}
```

Listing 3: Procedure '*Math.t0'* will get permission on *c* after adding '*claim* c'. '*Msth.c'* is assigned with permission values *1.0* in line *15*.

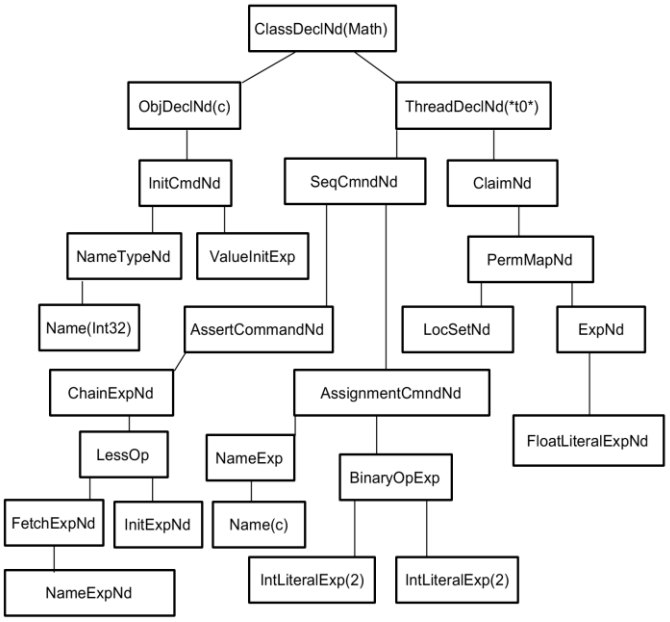

Figure 1: Abstract Syntax Tree (AST) generated by HARPO Compiler's parser from Listing 2

representation of AST of Listing 2 which is generated by the HARPO parser.

The Front-End of the verifier consists of generating AST, and, attributing the AST in checking phase. HARPO Parser is responsible for passing the legitimate syntax whereas, checker is responsible to check the names with their declarations, generating the symbol table, linking names to their declarations, and, detect the duplicate declarations. After resolution, symbol table is no more required, and we have a declaration list.

Back end of the verifier consists traversing the attributed AST stored as declaration list. An attributed AST, a declaration list, is show in Listing 4 is generated by HARPO verifier from Listing 2. This Declaration list is hunt down by Boogie backend for all declarations present in the list. Declarations may contain command and nodes. Each declaration and subsequent command or node are translated to its Boogie equivalent according to the information shown in Table 0.

```
[ClassDeclNd(
  ObjDeclNd[c](
    NamedTypeNd( Int32 ) : loc{Int32},
    ValueInitExpNd(IntLiteralExpNd(0):Int32):Int32),
  ThreadDeclNd[t#0](
    ThrdClaimNd( [ NameExpNd( c ) : Int32),
     SeqCommandNd(
      AssertCmdNd(
       ChainExpNd(
        [ LessOp ],
         [ FetchExpNd( NameExpNd( c ) : loc{Int32} )
: Int32, IntLiteralExpNd( 20 ) : Int32 ] ) : Bool )
) ) ) ) ),]
```

Listing 4: Abstract Syntax Tree (AST) declaration list generated by HARPO checker from Listing 2.

A string buffer containing Boogie prelude is appended with translated Boogie for each declaration in source program. String buffer moves around the subroutines, translating the declarations. When automated translation process is completed, string buffer contains Boogie code which is ready to verify with Boogie verifier.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have investigated the process of conversion of a program from HARPO to Boogie program. HARPO program specifications are written with the help of annotation. We automated the process of translation for '*assert'* and *'assume'* commands in HARPO, and it is checked with Boogie Verifier which is working according to our verification theory. Some commands like *if* and *while* have also been translated. Later, our implementation will result an independent backend of verifier, that would be able to translate complete HARPO programs which are exhibiting the concurrency among the threads. Some Language features, like functions and predicates, are needed to be added for writing full functional specifications. Finally, we are going to develop a verification tool like Dafny [11].

ACKNOWLEDGMENT

We thank Fatemeh Ghalehjoogh for developing verification theory for HARPO [7]. It would not be possible without her work to implement the verifier.

REFERENCES

[1] E. Dijkstra, "The humble programmer", Communications of the ACM, vol. 15, no. 10, pp. 859-866, 1972.
[2] T.S. Norvell, A.T. Md.Ashraful, L.Xiangwen, & Z. Dianyong, "HARPO/L: "A language for hardware/software codesign." in Newfoundland Electrical and Computer Engineering Conference (NECEC), 2008.
[3] T.S. Norvell, Language design for CGRA project. design 8. [unpublished draft], 2013.
[4] T. S. Norvell, "A grainless semantics for the HARPO/L language," in Canadian Electrical and Computer Engineering Conference, 2009.
[5] K.R.M. Leino, "This is Boogie 2," Microsoft Research, Tech. Rep., 2008, draft. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=147643
[6] O. Hasan, & S. Tahar. (2015). "Formal Verification Methods". In M. Khosrow-Pour (Ed.), Encyclopedia of Information Science and Technology, Third Edition (pp. 7162-7170). Hershey, PA: IGI Global. doi:10.4018/978-1-4666-5888-2.ch705
[7] Y.G. Fatemeh, "Verification of the HARPO language," Master's thesis, Memorial University, 2014.
[8] T. S. Norvell, Annotations for Verification of HARPOL. Draft Version 0. [unpublished draft], 2014.
[9] T. S. Norvell, "HARPO/L: "Concurrent Software Verification with Explicit Transfer of Permission" in Newfoundland Electrical and Computer Engineering Conference (NECEC), 2017.
[10] K.R.M. Leino, P. Müller, and J. Smans, "Verification of concurrent programs with Chalice," in Foundations of Security Analysis and Design V, ser. LNCS, vol. 5705, 2009.
[11] K.R.M. Leino & V. Wüstholz, (2014). The Dafny integrated development environment. arXiv preprint arXiv:1404.6602.