Formalizing Mathematical Developments to Support Verifying Compilers

Megan Bryant

Department of Mathematical Sciences
Clemson RSRG
Clemson University
Clemson, South Carolina 29634

`mrlebla@g.clemson.edu`

October 27th, 2013

Acknowledgements

I would like to thank my mentor, Dr. Murali Sitaraman from Clemson University for his continued support and guidance.

I would also like to thank the National Science Foundation. This research was funded in part by NSF grants DUE-0633506, DMS-0701187, CCF-0811748, DUE-1022941, and CCF-1161916 and Clemson University.

Outline

1. Verifying Compilers
2. Clemson RSRG
3. Math Theory Library
4. Areas of Future Research

A Grand Challenge

The creation of a verifying compiler is one of the current grand challenges in computing research [2].

A Grand Challenge

The creation of a verifying compiler is one of the current grand challenges in computing research [2].

Definition

A *grand challenge* is a long-range research goal whose resolution will have a significant impact on the field of research and society at large.

A Grand Challenge

The creation of a verifying compiler is one of the current grand challenges in computing research [2].

Definition

A *grand challenge* is a long-range research goal whose resolution will have a significant impact on the field of research and society at large.

Examples of other grand challenges:

A Grand Challenge

The creation of a verifying compiler is one of the current grand challenges in computing research [2].

Definition

A *grand challenge* is a long-range research goal whose resolution will have a significant impact on the field of research and society at large.

Examples of other grand challenges:

- P = NP

A Grand Challenge

The creation of a verifying compiler is one of the current grand challenges in computing research [2].

Definition

A *grand challenge* is a long-range research goal whose resolution will have a significant impact on the field of research and society at large.

Examples of other grand challenges:

- P = NP
- Cure for Cancer

A Grand Challenge

The creation of a verifying compiler is one of the current grand challenges in computing research [2].

Definition

A *grand challenge* is a long-range research goal whose resolution will have a significant impact on the field of research and society at large.

Examples of other grand challenges:

- P = NP
- Cure for Cancer
- Fermat's Last Theorem

Relevance

Why is the mathematical verification of software important?

Relevance

Why is the mathematical verification of software important?

Software bugs can have serious consequences of both a monetary and physical nature.

Relevance

Why is the mathematical verification of software important?

Software bugs can have serious consequences of both a monetary and physical nature.

- Estimated cost of errors in the U.S. alone is $60 billion per year.

Relevance

Why is the mathematical verification of software important?

Software bugs can have serious consequences of both a monetary and physical nature.

- Estimated cost of errors in the U.S. alone is $60 billion per year.
- Bugs in the software controlling vital hospital equipment have resulted in patient deaths.

Relevance

Why is the mathematical verification of software important?

Software bugs can have serious consequences of both a monetary and physical nature.

- Estimated cost of errors in the U.S. alone is $60 billion per year.
- Bugs in the software controlling vital hospital equipment have resulted in patient deaths.
- Miscalculations in the timing of missile defense systems have resulted in the destruction of military assets.

Relevance

Why is the mathematical verification of software important?

Software bugs can have serious consequences of both a monetary and physical nature.

- Estimated cost of errors in the U.S. alone is $60 billion per year.
- Bugs in the software controlling vital hospital equipment have resulted in patient deaths.
- Miscalculations in the timing of missile defense systems have resulted in the destruction of military assets.

The growth of large-scale software engineering will increase the costliness and frequency of these errors.

Traditional Testing Methods

What are some traditional software testing methods?

Traditional Testing Methods

What are some traditional software testing methods?

- Debugging by individual software engineers.

Traditional Testing Methods

What are some traditional software testing methods?

- Debugging by individual software engineers.
- Beta testing by interns (often unpaid).

Traditional Testing Methods

What are some traditional software testing methods?

- Debugging by individual software engineers.
- Beta testing by interns (often unpaid).
- Specialized commercial testing software.

Traditional Testing Methods

What are some traditional software testing methods?

- Debugging by individual software engineers.
- Beta testing by interns (often unpaid).
- Specialized commercial testing software.

Traditional methods are only able to test code under a subset of possible conditions and a subset of possible inputs.

Traditional Testing Methods

What are some traditional software testing methods?

- Debugging by individual software engineers.
- Beta testing by interns (often unpaid).
- Specialized commercial testing software.

Traditional methods are only able to test code under a subset of possible conditions and a subset of possible inputs.

They cannot guarantee the absence of errors.

Verifying Compilers

What exactly is a verifying compiler?

Verifying Compilers

What exactly is a verifying compiler?

Definition

A *verifying compiler* is a compiler which generates executable code and uses automated mathematical and logical reasoning to guarantee the correctness of that code to certain specifications.

Verifying Compilers

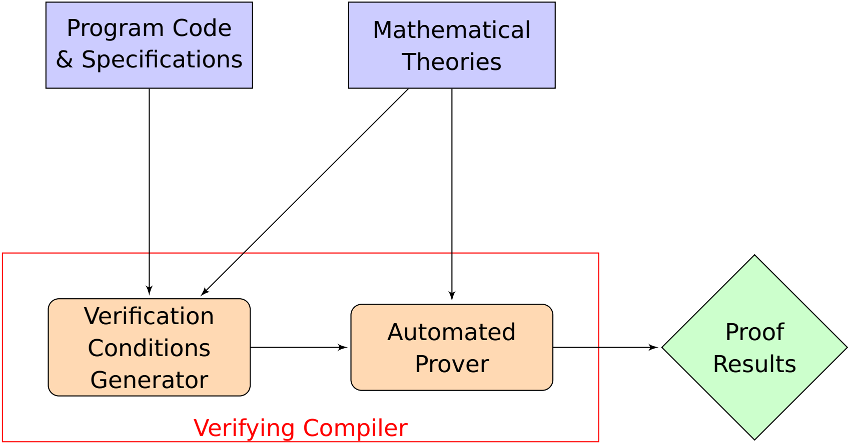
What exactly is a verifying compiler?

Definition

A *verifying compiler* is a compiler which generates executable code and uses automated mathematical and logical reasoning to guarantee the correctness of that code to certain specifications.

Verifying compilers prove with mathematical certainty the absence of errors in the generated code.

Verifying Compiler Overview



Program Code & Specifications

Mathematical Theories

Verification Conditions Generator

Automated Prover

Proof Results

Verifying Compiler

Clemson RSRG

Clemson RESOLVE Software Research Group is developing a push-button verifying compiler [1].

Clemson RSRG

Clemson RESOLVE Software Research Group is developing a push-button verifying compiler [1].

Definition

A *push-button verifying compiler* generates mathematical proofs of correctness and executable code in the same way regular compilers generate code [3].

Clemson RSRG

Clemson RESOLVE Software Research Group is developing a push-button verifying compiler [1].

Definition

A *push-button verifying compiler* generates mathematical proofs of correctness and executable code in the same way regular compilers generate code [3].

An integrated software language called RESOLVE is being developed to support the compiler.

Clemson RSRG

Clemson RESOLVE Software Research Group is developing a push-button verifying compiler [1].

Definition

A *push-button verifying compiler* generates mathematical proofs of correctness and executable code in the same way regular compilers generate code [3].

An integrated software language called RESOLVE is being developed to support the compiler.

Automated Prover

The RESOLVE verifying compiler depends on an automated prover to prove the generated verification conditions for each specification.

Automated Prover

The RESOLVE verifying compiler depends on an automated prover to prove the generated verification conditions for each specification.

Definition

Verification conditions are mathematical assertions which, when proven true, guarantee the correctness of pieces of code.

Automated Prover

The RESOLVE verifying compiler depends on an automated prover to prove the generated verification conditions for each specification.

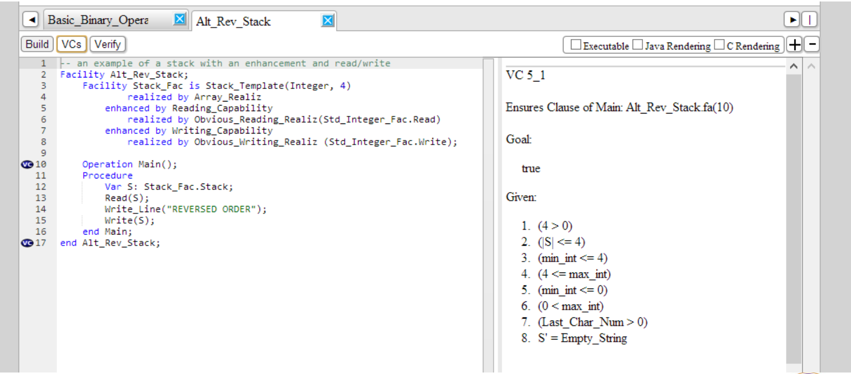
Definition

Verification conditions are mathematical assertions which, when proven true, guarantee the correctness of pieces of code.

The automated prover operates on a mathematical framework which relies on theory files stored in a coded math library.

RESOLVE Web IDE

The RESOLVE push-button verifying compiler utilizes a free web-based integrated development enviroment.



```
-- an example of a stack with an enhancement and read/write
Facility Alt_Rev_Stack;
    Facility Stack_Fac is Stack_Template(Integer, 4)
            realized by Array_Realiz
        enhanced by Reading_Capability
            realized by Obvious_Reading_Realiz(Std_Integer_Fac.Read)
        enhanced by Writing_Capability
            realized by Obvious_Writing_Realiz (Std_Integer_Fac.Write);

    Operation Main();
    Procedure
        Var S: Stack_Fac.Stack;
        Read(S);
        Write_Line("REVERSED ORDER");
        Write(S);
    end Main;
end Alt_Rev_Stack;
```

```
VC 5_1

Ensures Clause of Main: Alt_Rev_Stack.fa(10)

Goal:

    true

Given:

    1. (4 > 0)
    2. (|S| <= 4)
    3. (min_int <= 4)
    4. (4 <= max_int)
    5. (min_int <= 0)
    6. (0 < max_int)
    7. (Last_Char_Num > 0)
    8. S' = Empty_String
```

Mathematical Developments

Principal areas for the primary math library:

Mathematical Developments

Principal areas for the primary math library:

- Integer Theory

Mathematical Developments

Principal areas for the primary math library:

- Integer Theory
- Natural Number Theory

Mathematical Developments

Principal areas for the primary math library:

- Integer Theory
- Natural Number Theory
- String Theory

Mathematical Developments

Principal areas for the primary math library:

- Integer Theory
- Natural Number Theory
- String Theory
- Binary Relation Theory

Mathematical Developments

Principal areas for the primary math library:

- Integer Theory
- Natural Number Theory
- String Theory
- Binary Relation Theory
- Ordering Theory

Mathematical Developments

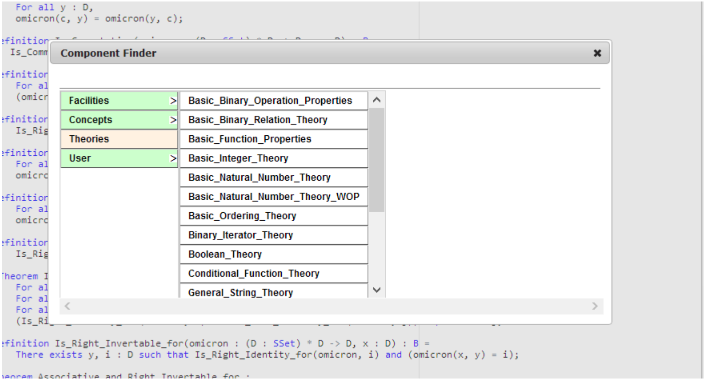
Principal areas for the primary math library:

- Integer Theory
- Natural Number Theory
- String Theory
- Binary Relation Theory
- Ordering Theory

With select theories, definitions, and properties from these areas, the automated prover is able to prove generated verification conditions.

Math Theory Library

The math theory library is integrated directly into the RESOLVE web IDE.



```
   For all y : D,
   omicron(c, y) = omicron(y, c);

efinition
  Is_Comm

efinitior
   For al
   (omicr

efinitior
   Is_Rig

efinitior
   For al
   omicrc

efinitior
   For al
   omicrc

efinitior
   Is_Rig

Theorem I
   For al
   For al
   For al
   (Is_Ri

efinition Is_Right_Invertable_for(omicron : (D : SSet) * D -> D, x : D) : B =
   There exists y, i : D such that Is_Right_Identity_for(omicron, i) and (omicron(x, y) = i);

eorem Associative and Right Invertable for :
```

RESOLVE Code

RESOLVE Code determining the maximum of two integers:

```
Facility Int_Max_Example_Facility;
Operation Max(restores I: Integer; restores J: Integer) : Integer;
ensures (Max = I or Max = J) and (Max >= I and Max >= J);
Procedure
Max := I + J;
If (I > J) then
Max := Max - J;
end;
If (J > I) then
Max := Max - I;
end;
end Max;
end Int_Max_Example_Facility;
```

Verification Condition Example

Example of a verification condition for the maximum integer program:

```
Goal:
((((I + J) - J) - I) = I or (((I + J) - J) - I) = J)

Given:
(min_int <= 0)
(0 < max_int)
(Last_Char_Num > 0)
(min_int <= J) and (J <= max_int)
(min_int <= I) and (I <= max_int)
(I > J)
(J > I)
```

Example Theory File

Excerpt from the basic binary operations theory file:

```
Precis Basic_Binary_Operation_Properties;
uses Boolean_Theory;

Definition Is_Associative(omicron : (D : SSet) ∗ D -> D): B =
For all x, y, z : D,
omicron(x, omicron(y,z)) = omicron(omicron(x, y), z);

Definition Is_Commutative(omicron : (D : SSet) ∗ D -> D, x : D) : B =
Is_Commutator_for(omicron, x);

Theorem I7: Is_Associative(+);

Theorem I10: Is_Commutative(+);
```

RESOLVE Translator: C

RESOLVE generated verified executable code for maximum integer program in C:

```
int Max(int  I, int  J){int Max= 0;
/*ensuresMaxIMaxJMaxIMaxJ*/

Max = I + J;
if(I > J){
Max = Max - J;
}
if(J > I){
Max = Max - I;}return Max; }
```

RESOLVE Translator: Java

RESOLVE generated verified executable code for maximum integer program in Java:

```
public static class Int_Max_Example_Facility{

public static int Max(int  I, int  J){int Max= 0;
/*ensures*/

Max = I + J;
if(I > J){
Max = Max - J;
}
if(J > I){
Max = Max - I;}return Max; } }
```

Integer Theory

RESOLVE code building the integers:

```
Precis Basic_Integer_Theory;
uses Monogenerator_Theory, Basic_Function_Properties,
Basic_Ordering_Theory, Basic_Natural_Number_Theory;

   Categorical Definition introduces Z: SSet,  0 : Z, NB : Z -> Z
    related by (Is_Monogeneric_for(Z, 0, NB));

    Definition 1 : Z = (suc(0));
    Corollary 1: For all m : Z, suc(m) = m + 1;
    Corollary 2: 1 : NN;
    Corollary 3: 4 not(=) 0;

    Theorem I15: For all m : Z, For all n : Z, -(m * n) = (-m) * n;
    Theorem I15: For all m : Z, For all n : Z, m * (-n) = (-m * n);
```

Natural Number Theory

RESOLVE code building the Natural Numbers using successor property:

```
Precis Basic_Natural_Number_Theory;
uses Basic_Binary_Operation_Properties,Basic_Ordering_Theory;

    Categorical Definition introduces N : SSet, 0 : N, suc : N -> N
    related by (Is_Monogeneric_for(N,0,suc));

    Definition 2: N = (suc(1));
    Definition 3: N = (suc(2));
    Definition 4: N = (suc(3));
    Definition 5: N = (suc(4));
    Definition 6: N = (suc(5));
    Definition 7: N = (suc(6));
    Definition 8: N = (suc(7));
    Definition 9: N = (suc(8));
```

Fundamental Theorems

In addition to basic properties and definitions, RESOLVE can code more complex theorems, including some famous examples⋯

```
Precis Major_Theorems;
uses Boolean_Theory, Set_Theory, Basic_Natural_Number_Theory;

Theorem Well_Ordering_Principle:
For all D : SSet,
D /= empty_set implies
(There exists min_element : D such that
(For all x : D, min_element <= x ));

Theorem Archimedean_Property:
(x : R and y: R and x >0)
implies (There exists n : N such that n > 0 and n*x > y );
```

Areas of Future Research

My immediate goal is to finalize the development of a math library which includes a full complement of basic theories and definitions.

Areas of Future Research

My immediate goal is to finalize the development of a math library which includes a full complement of basic theories and definitions.

I want to conduct an exhaustive test on the prover using the new math library and compare verification speed for different types of theorems.

Areas of Future Research

My immediate goal is to finalize the development of a math library which includes a full complement of basic theories and definitions.

I want to conduct an exhaustive test on the prover using the new math library and compare verification speed for different types of theorems.

The end goal is the full development of a successful verifying compiler and a resolution to the grand challenge.

References

[1] Resolve website http://http://www.cs.clemson.edu/resolve/.

[2] T. Hoare. The verifying compiler: A grand challenge for computing research. *J. ACM*, 50(1):63–69, Jan. 2003.

[3] M. Sitaraman, B. Adcock, J. Avigad, D. Bronish, P. Bucci, D. Frazier, H. M. Friedman, H. Harton, W. Heym, J. Kirschenbaum, J. Krone, H. Smith, and B. W. Weide. Building a push-button resolve verifier: Progress and challenges. *Form. Asp. Comput.*, 23(5):607–626, Sept. 2011.