# Formal Verification in Dafny: Use of Boogie Intermediate Verification Language for Determining the Correctness of a Program

Inaam Ahmed
Memorial University of Newfoundland
Email: inaama@mun.ca

## Abstract

Formal program verification has been used as a technique to ensure program correctness for several years. In this paper, the backend of formal verification system, named Boogie IVL, is discussed using interactive verifier named the Dafny. In this paper, the Dafny language and verifier are reviewed in detail. Some important features which are unavailable in high level programming languages like Java, C, and, Python but the Dafny language has direct support for those features namely sets, sequences, and, termination matric etc. The Dafny IDE and its integration available with Visual Studio is being used as verifier of a complete software system. It supports modularity, generic classes, abstraction, and, reverification. The Dafny compiler can produce both .NET executable code and verification conditions for Boogie Verification Debugger. We have also analysed the verification results of algorithms named, *bubble sort, $n^{th}$ fibonacci number,* and, *schorr waite* with benefits of using Dafny IDE. When the Dafny language was designed some essential techniques in formal verification namely, verification-splitting and timeout had not received attention by its developers. These features are highlighted in this paper, and we found that as compared to other verifiers e.g *VCC, Eiffel,* and, *Spec#* the Dafny verifier IDE has more interactive program verification system.

**Keywords:** Formal Verification, Dafny Verifier, SMT solver, Boogie IVL, BVD

## 1 Introduction

The purpose of program verification is to provide an error free software system. A program testing with random sets of inputs can show the presence of hidden errors but not their absence [1]. A critical software system needs its mathematical accuracy guaranteed before launching the system. In formal program verification, computer programs are considered as mathematical objects and their properties are verified by mathematical proofs. Ultimately formal methods and mathematical techniques are used to verify software-specifications to provide error free software. Some computer-based software uses simulation and testing principles, however the exponential increase in test patterns during the testing process increases the duration of program testing. The testing and simulation cannot verify precisely the properties about the continuous systems due to floating point and fixed-point representation of real numbers in computers. Testing has become insufficient to prove that system is error free [1]. This technique of using the formal verification method is primarily based on fundamentals theory of automata in theoretical computer science. The automata theory has strictly typed language constructs and logic calculi that can handle complex systems with less chance of human errors.

The Program verification can also be performed with the help model checking, or symbolic verification [2]. In this paper, theorem proving methods are discussed which are primarily used in the Dafny verifier. This mathematical methodology of the program verification proves accuracy of programs hundred percent by using theorem provers [2]. The theorem provers usually consist of well known axioms and primitive - inference rules. The Language like C++ or Java has not sufficiently qualified to carry out theorem proving, because the semantics of these high-level languages are ambiguous, and this ambiguity results in more than one interpretations. These languages can not be used in theorem proving where we need a logical language primitive and syntax that can be described using unambiguously defined rules and semantics.

The Dafny tool fulfills need of the formal verification using a programming style named ‘design-by-contract’. It is easy to learn and incorporates many features of good programming language which are based on imperative programming paradigm. During a program verification process, states and properties are checked at the entry and exit of a method[1] in a program. The Dafny’s verification methodology is defined in terms of the Boogie in its backend [3], whereas the annotations in the Dafny are written in a format that is like a high-level programming language one. It converts the specifications into an intermediate verification language named Boogie Intermediate Verification Language[2] (Boogie IVL) [3, 5, 6]. There are numerous verification tools except the Dafny have been designed for past few years based on the Boogie’s verification methodology including VCC, Eiffel, and Spec# [3]. The Boogie generates first order verification conditions for given specifications. These verification conditions are given onward to SMT solver for checking the correctness of these conditions. The Boogie verification debugger does provide verification error information in detail that can be used to get the right values of variables [4].

## 2 Boogie IVL

This is a common practice in a verification process to convert a given program into an intermediate verification language which is more structured like program than formulas. The verification of a program is performed in the Boogie IVL transforming that single program into set of verification conditions [4]. Furthermore, logical formulas are used to test the verification conditions, their correctness determines the correctness of the given program. Some verifiers namely Spec#, C, Dafny, Java bytecode with BML, and Eiffel are languages using the Boogie as intermediate verification language in their verification function [4]. In sections 7 and 8, we discussed the Dafny’s advancements and advantages with its limitations.

[1] Method is an independent piece of code in any programming language. In this paper, method is used with the same meaning as methods in high-level programming language.

[2] The Boogie IVL and Boogie are used interchangeably in this paper.

## 3 Verification Using Dafny

There are two extremes of the formal program verification system, on one side we try to find only the bugs in a program on the other side we try to ensure the functional correctness of a program is proved [2]. The Dafny verification provides user defined function and ghost variable to verify the modular programs. Usually modules are verified separately and the Dafny verifier only knows about the interfaces of modules visible to their caller modules [5].

Since backend of the Dafny is dependent on the Boogie IVL then the results of the Boogie imply the results of verification in the Dafny. Now, the Dafny has become a general-purpose tool for verification of program specifications. The program verification using the Dafny takes less time and it responds timeout message after ten seconds in case of missing proof [3]. The verification of *Schorr-Waite* algorithm using Dafny in fact, its entire program text (including its full functional correctness) took less than five seconds [5] which is significantly smaller then the program testing platforms and model checkers.

## 4 Language Features of Dafny

The Dafny has smart features which are directly unavailable in an imperative high-level programming language. Some important constructs of the Dafny are given below:

### 4.1 Types

The Dafny has types including boolean, integer, reference to object, sets, sequences, and, user-defined data types except the sub typing feature. The sets are used to describe the read/write frames of program, whereas algebraic types are used for writing specifications of a program [5].

### 4.2 Pre and Post Conditions

The Dafny uses precondition and postconditions to check the correctness of a module where pre and post condition are declared. A Precondition is provided with the keyword ‘**requires’** followed by an expression and postcondition is given with keyword **ensures** followed with a boolean expression. We can write more than one pre and post conditions for a given program when we write its specification. A failure to satisfy any of the condition(pre/post) will give error during the verification. The caller of a method should provide satisfactory precondition and same method must come up with results satisfying post condition to guarantee its correctness. The execution of calling function will change the program state which in turn update the program variables [5]. The Parts of program or memory locations(variables) can be changed during method execution. These set of locations are defined as frame of a method [7].

### 4.3 Ghost States

The variables in the Dafny language can be marked as **ghost** fields. The ghost variables are omitted by compiler and do not show up into an executable code. Their values can not be assigned to actual variables. As long as it is concerned to verification, ghost variables and actual variables are same. The Ghost variables are very handy when writing the preconditions and the post conditions [6].

### 4.4 Modification

The variables in a program can be read from and written into when executed. Dafny checks whether system can read or write the variable using **modify** clause [10]. Modify clause declare the frame of module defining which locations are accessible by the part of a program. The Granularity of framing in the

Dafny is slightly different: instead of defining frame with variables the Dafny define frames having permission to object. The Permission transfer is also an important concept here to understand more deeply the mechanism of frame accessibility. These permissions are transferred from a *caller* module to a *callee* and back from *callee* to *caller* having the old ancient analogy called passing the baton. The person who has stick can talk and others will listen afterwards he should give stick to someone else in the crowd and same protocol applied repeatedly. Similarly, during concurrent programs verification permission transfer scenario must be maintained in a way of passing the baton. A part of a program having access permission to memory frame can access the location. The Behavior of permission transfer is explained in [7] using different examples.

### 4.5 Function

A function can be declared inside class. By default, functions in the Dafny are ghost code, they are implemented in an executable program code; However, functions declared with methods (*function method*) in a specific way can be used into a compiled code. Function has type parameter function body and required annotations (*read, modify, ensures, requires, and decreases*) [10].

### 4.6 Sets

The Dafny language has provided finite sets declaration. The normal set operations are applicable except cardinality and complement. The set operations are converted into the Boogie operations while converting into, e.g. *A=B* will be converted to *setEqual(A,B)* [6].

### 4.7 Algebraic Sequences

The Dafny has support of sequence declaration. The Operations on these sequences include concatenation, selection and finding length of sequence [6]. Application of quantification is not simple as to deal with bound variables, operations with sequence index involved are not flexible to quantification.

### 4.8 Termination

The Loops are declared with loop invariant. An invariant of loop must be true during all iterations of the loop. For recording the iterations, the Dafny uses **decreases** clause to check the iterations will not go infinite [10]. A problem with this termination technique is to give a loop guard as decreasing valued expression [6].

## 5 Dafny IDE

In past several years, an interactive verifier has become a concern of the software developers to prove the correctness of a programs. These verifiers guarantee verified software to their beta[3] users and hence customers. The verification of a program manually is prone to error and takes long time to traverse through all states of program without modularity. However, the modular programs, one verification attempt does not determine the correctness of next turn still checking the same module again using manual verification. The Dafny IDE provide an attractive approach used to verify programs in background. The Dafny verifier work as a component of the Dafny compiler. The Dafny compiler can generate *.exe* code from given annotations as given in Figure 1.

[3] Term used to denote the field testing before launching the system.

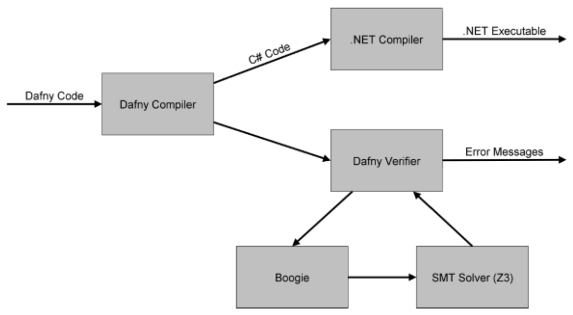

*Figure 1 : The Dafny system composed of the Dafny verifier and .NET compiler. The Illustration is taken from [6].*

The Dafny verifier was designed to deal with the modular programs in a way to speed up the verification process by eliminating the modules which are unnecessary verify and still waiting in verification queue [3]. A modular program verification is achieved by implicit dynamic frames [7]. In [3] five snapshots of same program with minor changes are verified with small increment of verification time than verification of one program due to modular verification compatibility. Table 1 shows the numerical values:

*Table 1: Every program is verified by running 5 time and running once with 5 snapshots gives very less increase in verification duration [3].*

| Program | 3 solver instances (5 runs) | 3 solver instances (5 snapshots) |
|---|---|---|
| `LnSystemF.dfy` | 510 $(5*102)$ | 123 |
| `ParallelBuilds.dfy` | 1345 $(5*269)$ | 311 |

The Dafny uses multithreading to perform verification of different modules at the same time. An essential feature of Dafny is to report error in program with every possible information of the program trace where errors are present [3]. The Dafny IDE does not only provide information about error but suggestions to remove that error by providing hover choices.

Recently the Dafny tool architecture has been updated with integration of the Boogie Verification Debugger (BVD) as shown in Figure 2. The verifications of different entities are isolated, as a result reverification becomes faster and it can be completed in less time if program was not updated.

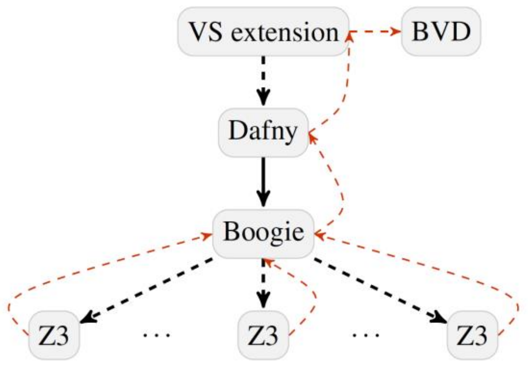

*Figure 2: Current tool architecture of Dafny IDE, black dotted arrows showing flow of program data and red dotted arrows show the error reporting direction. The figure taken from [3].*

## 6 Verified Software Construction

Since the formal verification ensures the correctness of program. Previously we described the design of Dafny and its IDE's integration with Visual Studio; however, it is always a great concern of software developers to ensure the correctness of complete software. A large software system is verified before giving it into the hands of user (personal or commercial). Dramatic consequences of the errors in software systems have served as an incentive towards the mathematical software designs and growth in technologies based on formal methods [5]. The Software designed for an aerospace, avionic system, railway station, or medical devices is needing to be ensured about correct mathematically. In previous ten to fifteen years serious damages caused by software failures results into greater financial damages and this software defected the system availability [1].

The Dafny is also used to ensure the reliability of the software. The verification of complete software systems is a tedious task. Programmers design specifications of program relatively familiar with Hoare Logic proposed in seminal paper by C.A.R Hoare in 1969 [8]. The verification tools provide help to detect the presence and absence of the bugs which results in high quality reliable software. The development in an automated verification system is research area of software development providing help to create more practical tools. A well known *"bubble sort"* algorithm has verified by using the Dafny IDE with successful verification of its code [9]. The Dafny verify software module by module. Its IDE helps to write the specifications for complete program and specifications editing platform assists programmer to find right the specification. After writing specification Dafny IDE allows to verify the specification by converting the given program into intermediate boogie equivalent and output will be given to Boogie verification tool integrated with Dafny. This underlaying Boogie tool will provide program to SMT solver named Z3. Process of this verification is described in Figure 2.

## 7 Conclusion

Verification of a program is essential before it is launched or used in an operating environment. Following are some key points about formal software verification:

- The Formal verification is essential for software systems to run without error [1].
- The verification tools that can be used to verify the reliability of software namely Dafny, Spec#, VCC and Eiffel etc.
- The Dafny is popular among formal verification tools due to its IDE integration with Visual Studio.
- The Dafny infer the termination and ensures it, but sometimes programmer has to provide decrease clause explicitly.
- The Dafny IDE can be used for checking its correctness of a large software system.

## 8 Limitations

- The type-casting feature is not available in the Dafny.
- The formal verification tools are dependent only on the Boogie.
- The Dafny does not take advantage of splitting for selective verification of the program parts.
- The verification time out in the Dafny should be smaller, because in case of missing proof a longer time interval will not work.
- There is not a way to show information in middle of the Dafny's verification process.

## 9 Future Work

In future work, we are intended to build compiler that generate executable code from specifications of program.

## 10 References

[1] Myers, G. J., Sandler, C., & Badgett, T. (2011). The art of software testing. John Wiley & Sons.

[2] Hasan, O., & Tahar, S. (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

[3] Leino, K. R. M., & Wüstholz, V. (2014). The Dafny integrated development environment. arXiv preprint arXiv:1404.6602.

[4] Leino, K. R. M. (2008). This is boogie 2. Manuscript KRML, 178(131).

[5] Leino, K. R. M. (2010, April). Dafny: An automatic program verifier for functional correctness. In International Conference on Logic for Programming Artificial Intelligence and Reasoning (pp. 348-370). Springer, Berlin, Heidelberg.

[6] Herbert, L., Leino, K. R. M., & Quaresma, J. (2012). Using Dafny, an automatic program verifier. In Tools for Practical Software Verification (pp. 156-181). Springer, Berlin, Heidelberg.

[7] TS. Norvell "Concurrent Software Verification with Transfer of Permissions.", in Proc. The Twenty-Sixth Annual Newfoundland Electrical and Computer Engineering Conference", November 15th, 2017, Newfoundland

[8] Hoare, C. A. R. (1969). An axiomatic basis for computer programming. Communications of the ACM, 12(10), 576-580.

[9] Lucio, P. (2017). A Tutorial on Using Dafny to Construct Verified Software. arXiv preprint arXiv:1701.04481.

[10] Ford, R. L., & Leino, K. R. M. Draft Dafny Reference Manual.