Automated Theorem Proving & Applications

(Review)

Thomas Swed
Lewis University
1 University Pkwy,
Romeoville, IL 60446
thomasdswed@lewisu.edu

Abstract—**This paper gives a summary of the concepts presented in "Automating Theorem Proving with SMT" [1] and "Problem-Oriented Applications of Automated Theorem Proving" [2]. In the first article, specific characteristics and uses of an SMT solver tool called Dafny are described. Several examples and proofs are given to convey the capabilities and automation possible with this tool. The latter article concentrates on implementing an Automated Theorem Proving system and account for challenges faced with differing types of logic.**

Keywords—Automated Theorem Proving, Mathematical Proof, Dafny, Recursion, Co-datatype, Lemma, Induction, Co-induction, Co-recursion

I. Introduction

Satisfiability-modulo theories (SMT) solvers exist to verify whether a first-order logic formula is satisfiable. Their functionality continues to improve and evolve while they continue to provide a high degree of automation. Numerous verification tools use SMT solvers as foundation to automate tasks. While SMT solvers can be applied to many fields, this paper will examine Dafny as an example of a verification tool used to evaluate computer programs specifically. It utilizes an SMT solver as its fundamental system and supports several proof features including induction and co-induction [3]. Examples are given to show the capabilities of these features.

Automated theorem proving is used when a given problem can be formalized in a specific type of mathematical logic utilizing proof calculus. In many cases this is not possible and therefore must use a different type of automated verification. Challenges faced with developing a system to interact with classical, intuitionistic and modal logics as well as input transformations, inference machines and proof presentations are covered.

II. Key Terms

- A *Co-Inductive Datatype* is a datatype in the Dafny language whose constructors are evaluated lazily and allow infinite structures (lists) [4]. It is declared using the keyword *codatatype.*

- *Recursion* occurs when a method invokes or calls itself. It is used to solve a problem by solving smaller instances of the same problem.

- *Co-recursion* corresponds to recursion in that while recursion breaks data down into smaller pieces until it reaches a base case, co-recursion starts from a base case and builds up to produce further data [5]. This allows programs to produce potentially infinite data structures.

- *Mathematical Induction* is a proof technique used to prove a given statement about a well-founded structure. The method involves proving an initial statement formally in order to prove that a second statement is true [6].

- *Co-Induction* defines infinite data structures known as *codata*. It is used to show that an equation is satisfied by all possible implementations of such a specification [7].

- In mathematics, a *lemma* is a proved proposition used as a stepping stone to a larger result [8]. In the Dafny language it is represented as a method that evaluates a pre and post condition.

- *Theorem proving* utilizes mathematics and different types of logic to accomplish the task of showing that a statement is true based on previously established statements. It does not need to check a programs state in order to verify properties [9]. While in many cases, this is a labor-intensive, manual task; automated theorem proving can be used in certain instances.

III. Dafny

Dafny is a programming language with a built-in program verifier. It is designed to support the static verification of programs [10]. The next section will cover examples to demonstrate how Dafny implements the proof features of induction, co-induction and declarative calculations.

A. Induction

In Dafny, a lemma is expressed as a method that terminates in a state satisfying the post-condition when any value of the method parameters satisfies the method's precondition [11]. It is expressed as follows [12]:

```
Ghost method Tail_ Lemma(s: Stream, n: int)
        Requires 0 <= n;
        Ensures Tail(s, n) = Tail_Alt(s, n);
```

The precondition is represented by the keyword *requires* while the *ensures* keyword signifies the post-condition, what is being proved. The *ghost* keyword is used as a flag for Dafny's compiler to let it know that what follows is only to be used by the verifier. The verifier proves this lemma when a method body is provided to evaluate if the post-condition has been met.

B. Co-induction

Co-predicates are used in Dafny to define a property of a co-inductive datatype. The following method is given as an example [13].

```
copredicate AllEven(s: Stream<int>)
{
  s.head % 2 = 0 ^ AllEven(s.tail)
}
```

Co-predicates are defined as the greatest solutions of the recursive equations to which their definitions give rise [14]. They can be used to define and outline code but are not part of the implementation of code.

C. Summary

Dafny enables a vast amount of automated formal proof capabilities by using an SMT solver. It minimizes human interaction and encapsulates details for ease of use. The tool has high performance times giving instant results and can be run in the background of a development environment.

IV. Automated Theorem Proving

Several challenges are faced when developing an ATP system. ATP is typically formalized using classical first order logic (FOL). One challenged faced is that many applications use different types of formalized logic as FOL is not well suited to cope with the demands of some systems [15]. Another challenge relates to the highly technical nature of the mathematical proofs generated by the ATP system. Because of their complex nature, the proofs must be converted to a form that is understandable to the average user.

An ATP system is presented with personalized examples to account for three different types of applications: viz. mathematics, programming and planning [16]. Corresponding to each application, a specific logic is used. Classical, intuitionist, and modal logic, respectively. Biebel et al

- *Classical logic* allows the most flexibility for ATP systems as it provides the capability of a normal form for formulas [17]. It deals with proving theorems mathematically and allows input to be transformed into normal form or non-normal form.
- *Intuitionistic logic* applies to applications that utilize constructive proofs and does not exist in a normal form, preventing STP systems to be used extensively. However, the non-classical features in the logical formulae may be regarded as semantic information in addition to the formulae's logical contents in the classical sense [18].
- *Modal logics* are used in applications that apply epistemological concepts [19]. It is similar to intuitionistic logic in that a normal form does not exist but efforts are being made to interact with semantical meta-knowledge.

V. Research Project Application

For my research project, I will be researching various types and applications of different databases to gain a better understanding and determine which system will be the most efficient to use in my current role as an analyst. The characteristics, strengths and weaknesses of traditional relational databases as well as distributed systems will be evaluated. It is quite possible that there is not a clear-cut answer to the question of which database to use. There are many different database platforms and types as there are differing needs in a given organization. Perhaps the solution is to apply multiple databases when needed to optimize performance and customize the system to solve different problems.

There are a number of variables to consider when evaluating a database system. Some of which include: the number of users, the number of dedicated servers for the system, complexity of the data being stored, replication and security factors.

- The *number of users* is particularly important because it will determine the load placed on particular servers running the database. Things like the complexity of queries, type of information and hardware components must be considered and optimized.
- The *number of servers* available and hardware capabilities of each server will influence the performance times as well as data replication and backup.
- What *type of data* needs to be stored and queried will be a major factor in determining what type of database to use. This could be a combination of differing databases depending on the needs of a given system. There are also several variables to consider here: volume of data, consistency of data, complexity of data, query demands, and visualization needs. Incompatible data formats, incomplete data, non-aligned data structures, and inconsistent data semantics represent significant challenges when interacting and querying data.

- *Database replication* consists of duplicating data operations from a primary database to a secondary database. Challenges are encountered when addressing issues of securing the data, synchronization, and performing real-time data movement while maintaining reliable and flexible performance speeds.

- *Security* is a general concern for an entire company's internal network, but especially when relating to a database which potentially houses sensitive information.

There are several benefits of applying an ATP system to design and optimize a database. Verification will provide a greater amount of coverage than testing or debugging, as well as identify errors early in the process. An automated testing system means less human interaction and thus, less chance of errors in a manual step-by-step process. It can also locate errors earlier than debugging while providing greater code coverage.

VI. Conclusion

ATP focuses on developing deductive systems which simulate mathematical reasoning to verify a given system's specification [20]. Traditionally, while Automated Theorem Proving has been a very efficient verification tool to use in many cases, it comes with a steep learning curve and labor-intensive process. The research presented in [2] has given examples of simplifying the technical nature of these tools to encapsulate technicalities from users while maintaining functionality and efficiency. Reference [1] summarized the verification tool Dafny and its applications. Examples were offered to demonstrate inductive and co-inductive proof definitions. The level of automation achievable with ATP systems provides efficient, comprehensive verification to users of these systems.

References

[1] Leino, K.R.M.: Automating theorem proving with SMT. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 2–16. Springer, Heidelberg, 2013.

[2] W. Bibel, D. Korn, C. Kreitz, S. Schmitt, Article: Problem-Oriented Applications of Automated Theorem Proving, . Design and Implementation of Symbolic Computation Systems, LNCS 1126, pp. 1-21, 1996.

[3] Leino, 2.

[4] Richard L. Ford, K. Rustan M. Leino, Article: Draft Dafny Reference Manual, January 2016, pp. 91.

[5] Corecursion, Wikipedia, https://en.wikipedia.org/wiki/Corecursion. Accessed on May 3, 2017.

[6] Coinduction, Widipedia, https://en.wikipedia.org/wiki/Coinduction. Accessed May 3, 2017.

[7] Lemma (mathematics), Wikipedia, https://en.wikipedia.org/wiki/Lemma_(mathematics). Accessed May 3, 2017.

[8] Mathematical Induction, Wikipedia, https://en.wikipedia.org/wiki/Mathematical_induction. Accessed May 3, 2017.

[9] Thomas Reinbacher, Article: Introduction to Embedded Software Verification. Embedded Computing Systems Group, Vienna University of Technology, Austria, 2008, 3.

[10] Microsoft, Dafny: A Language and Program Verifier for Functional Correctness, https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/. Accessed May 2, 2017.

[11] Ibid.

[12] Leino, 3.

[13] Leino, 3.

[14] Leino, 6.

[15] Bibel, 2.

[16] Bibel, 2.

[17] Bibel, 2.

[18] Bibel, 2.

[19] Bibel, 2.

[20] Bibel, 1.