A Method to Deduce and Synthesize the Dafny Programs

WANG Changjing¹,², DING Xilong¹, HE Jiangfei¹, CHEN Xi¹, HUANG Qing¹, LUO Haimei³, ZUO Zhengkang⁠†¹

1. College of Computer Information Engineering, Jiangxi Normal University, Nanchang 330022, Jiangxi, China;
2. Management Science and Engineering Research Center, Jiangxi Normal University, Nanchang 330022, Jiangxi, China;
3. College of Physics and Communication Electronics, Jiangxi Normal University, Nanchang 330022, Jiangxi, China

© Wuhan University 2021

Abstract: We propose a systematic method to deduce and synthesize the Dafny programs. First, the specification of problem is described in strict mathematical language. Then, the derivation process uses program specification transformation technology to perform equivalent transformation. Furthermore, Dafny program is synthesized through the obtained recursive relationship and loop invariants. Finally, the functional correctness of Dafny program is automatically verified by Dafny verifier or online tool. Through this method, we deduce and synthesize Dafny programs for many typical problems such as the cube sum problem, the minimum (or maximum) contiguous subarray problem, several searching problems, several sorting problems, and so on. Due to space limitation, we only illustrate the development process of Dafny programs for two typical problems: the minimum contiguous subarray problem and the new local bubble sorting problem. It proves that our method can effectively improve the correctness and reliability of Dafny program developed. What’s more, we demonstrate the potential of the deductive synthesis method by developing a new local bubble sorting program.

Key words: Dafny; deductive synthesis; specification transformation technology; recursive relationships; loop invariants

CLC number: TP 305

Received date: 2021-10-16
Foundation item: Supported by the National Natural Science Foundation of China (61762049, 61862033, 61902162, 11804133) and Natural Science Foundation of Jiangxi Province (20202BAB202025, 20202BAB202026, 20202BAH202015)
Biography: WANG Changjing, male, Ph. D., Professor, research direction: software formal method, trustworthy software. E-mail: wcj771006@163.com
† To whom correspondence should be addressed. E-mail: zhengkang2005@iscas.ac.cn

0 Introduction

From the perspective of computer science, it is almost impossible to create completely reliable software except for very small programs[1]. We have known for decades about the principles and techniques for ensuring that programs are correct, but it is not easy to verify the program correctly. As verification tools become powerful and practical in the past decade, it makes formal software verification feasible. For example, in order to improve the efficiency and reliability of software development, Microsoft recruits experts and outstanding graduates to specialize in the research of formal methods and verification tools. At the same time, with the rapid development of automatic proof theory, the improvement of CPU computing power, and the enhancement of theorem prover, the program proof tasks emerge in large numbers, and many well-designed program provers appear[2].

With the development of formal verification technology, the successful application of formal verification tools in program verification continues to increase. Due to the advantages of formal methods[3] in the development of complex and reliable software, formal verification tools will significantly improve our efficiency in verifying reliable algorithm software in the future. In addition, formal verification tools have the following advantages: higher verification efficiency and no need to write certificates manually. Dafny, one of the most representative formal verification tools, can verify the correctness of the program and provide complete automation, and also can reduce the workload of our comprehensive verification of the program[4].

This paper uses formal method to develop Dafny programs for two typical problems: the minimum contiguous subarray problem and the new local bubble sort-
ing problem. The final developed programs are automatically verified by Dafny verifier. The main contributions are as follows:

1) We propose a system method to deduce and synthesize the Dafny programs. Initially, the specification of problem is described in strict mathematical language. Then, the derivation process uses program specification transformation technology to perform equivalent transformation. Furthermore, Dafny program is synthesized through the obtained recursive relationship and loop invariants. Finally, the functional correctness of Dafny program is automatically verified by Dafny verifier or online tool.

2) We deduce and synthesize Dafny programs for many typical problems such as the cube sum problem, the minimum (or maximum) contiguous subarray problems, several searching problems, several sorting problems. Due to space limitation, we only illustrate the development process of Dafny by focusing on two typical problems: the minimum contiguous subarray problem and the new local bubble sorting problem. It proves that our method can effectively improve the correctness and reliability of the developed Dafny program.

3) We demonstrate the potential of the deductive synthesis method by developing a new Local Bubble Sorting program.

The organizational structure of this article is as follows: Section 1 is the introduction of related work; Section 2 is about related technologies and tool, briefly introducing the specification transformation technology and Dafny; Section 3 is the deductive synthesis Dafny programs for minimum contiguous subarray problem and bubble sorting problem; Section 4 is conclusion and future work.

1 Related Work

In this section, we survey prior work that are closely related to the method proposed in this paper.

Program synthesis. There has been a great deal of interest in automated synthesizing programs from high-level expressions of user intent [5-7] in the past decade. Some of these techniques are geared towards computer end-users and they utilize informal specifications such as input-output examples, natural language, or a combination of both. On the other hand, program synthesis techniques are geared towards programmers who often utilize additional information, such as a program sketch or types in addition to test cases or logical specifications. Most of the program synthesis methods are obtained from semi-formal description (such as UML) or informal description (such as natural language). The main difference between those methods and ours is that we deduce the Dafny program from formal specification, which is helpful to subsequent formal reasoning. In addition, our method synthesizes recursive relationships and loop invariants to develop the Dafny programs.

Program transformation. Program transformation was introduced by researchers 50 years ago, and then it was formalized[8-10]. From then on, program transformation technology has gradually developed. Program transformation converts one program into another[10]. Learning the idea of program transformation, we design the specification transformation technology. Program transformation converts an initially inefficient program into an equivalent program. Our method is different from program transformation by generating an effective algorithm from the problem specification. We also learn from the idea of transformation strategies[11-13] to get better transform rules.

Program refinement[14]. At present, some good program refinement work has been applied to practice. The authors of Refs.[15-17] proposed a mechanism to gradually refine the specification into executable code by introducing implementation details. In our method, we use strict formal derivation technology and specification transformation technology to make the final derivation procedure more rigorous and correct.

2 Related Technology and Tool

In this section, we mainly introduce program specification transformation technology and Dafny.

2.1 Program Specification Transformation Technology

The proof of correctness of program, formal derivation and functions depends on whether the specification is correct and whether the construction is proper either to a great extent. Some quantifiers should be used when describing the program specification of the problem[18]. The general form is: \( (Q:i: f(i)) \), \( r(i) \) is the range of the constraint variable \( i \), and \( f(i) \) is the function of the constraint variable. The meaning of this form is “the quantity obtained by performing \( q \) operation on the function \( f(i) \) in the range of \( r(i) \)”. The quantifier \( Q \) mainly includes \( \forall \) (universal quantifier), \( \exists \) (existential quantifier), \( \Sigma \) (summation quantifier), \( \Pi \) (quadrature quantifier),
These features allow modular verification\cite{21} of the functions, algebraic data types, sets, and sequences. Dafny uses a syntax similar to other programming languages, but it requires some special tools to help with formal verification. Annotation is a tool provided by Dafny to help users verify the program. Annotation is not part of the actual program, but it provides specifications and information about methods in the program. By considering the annotations in the function, Dafny can more directly verify the correctness of the program. Dafny transforms the burden of writing bug-free code into the burden of writing bug-free annotations. It is usually easier than writing code, because annotations are shorter and more direct. In addition, the behavior of writing annotations can help people understand what the code does at a deeper level. Dafny can also prove that there are no runtime errors, such as index out of bounds, null cancellation, division by zero, etc.

An overview of the entire Dafny system is given in Fig. 1. The way programmers interact with it is the same as static type checker, when the tool reports an error, the programmer will respond by changing the program’s type declaration, specifications, and statements. After the Dafny code passes the Dafny compiler, there are two paths: one is that the Dafny compiler directly generates C# code, and then compiles it to the MSIL byte code of the .NET platform; the other is that Dafny’s program verifier converts the given Dafny program into the intermediate verification language Boogie\cite{22}. Then, the Boogie tool generates a first-order verification condition, and finally passes it to the Z3\cite{23} SMT solver\cite{24} for verification. Any content that violates these conditions will be returned as a verification error.

<table>
<thead>
<tr>
<th>Table 1  Keywords table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ensures boolean expression (postconditions)</td>
</tr>
<tr>
<td>assert boolean expression</td>
</tr>
<tr>
<td>invariant boolean expression (that holds before during, and at the conclusion of a while loop)</td>
</tr>
<tr>
<td>decreases ranking function</td>
</tr>
<tr>
<td>forall counter :: range ⇒ boolean expression</td>
</tr>
<tr>
<td>reads immutable object</td>
</tr>
<tr>
<td>modifies mutable object</td>
</tr>
</tbody>
</table>

3 Deductive Synthesis of Dafny Programs for Two Typical Problems

In this section, we will use the deductive and synthesis method to develop Dafny programs\cite{25} for two typical problems. One is the minimum contiguous subarray problem; another is the new local bubble sorting problem.

3.1 Minimum Contiguous Subarray

Problem description: Calculate the minimum sum of
adjacent elements in a given integer array $a[0: n-1]$.

- Provide problem specification
  Using minsum($n-1$) to represent the smallest sum of contiguous subarray of its input array $a[0: n-1]$.

$Q$: $n \geq 0$

$R$: minsum($n-1$) = (MIN $i, j: 0 \leq i \leq j < n$: sum($i, j$))

The auxiliary function sum ($i, j$) is defined as follows:

$$\text{sum}(i, j) = \sum_{k=i}^{j} a[k] = \text{sum}(i, j-1) + a[j-1], \quad i \leq j$$

sum($i, j$) = 0, if $i > j$

The auxiliary function min is defined as follows:

$$\text{min}(a, b) = \text{if } a > b \text{ then } b \text{ else } a$$

- Partition the original problem
  Divide the original problem into sub-problems with the same structure, then construct a recursive relationship for solving the problem and a preliminary algorithm.

Obviously, the most commonly used method for this problem is the exhaustive method. That is to list the sum of all adjacent elements and compare them to get the solution of the minimum contiguous subarray program, and the time complexity of the algorithm is $O(n^3)$. But, we derive another algorithm to reduce the time complexity and optimize the algorithm. We divide the original problem as follows:

$$\text{minsum}(a[0:n-1]) = F(\text{minsum}(a[0:m-2], a[m-1]), 0 \leq m \leq n)$$

We start with the post-assertion to find the recursive relationship $F$.

- Find the recurrence relations
  minsum($a[0:m]$) =
  $$(\text{MIN } i, j: 0 \leq i \leq j \leq m: \text{sum}(i, j))$$
  $= [\text{cross-product nature}]$
  $$(\text{MIN } j: 0 \leq j \leq m: (\text{MIN } i: 0 \leq i \leq j: \text{sum}(i, j)))$$
  $= (\text{MIN } j: 0 \leq j \leq m: \text{ms}(j))$

Define $\text{ms}(j) = (\text{MIN } i: 0 \leq i \leq j: \text{sum}(i, j))$

$$= \text{min}(\text{MIN } j: 0 \leq j \leq m: \text{ms}(j)), \text{ms}(m))$$

$= [\text{range division and single range}]$

$$= \text{min}(\text{minsum}(a[0-m-1]), \text{ms}(m))$$

The first recursive relationship is:

Recurrence 1:

$$\text{minsum}(a[0:m]) = \text{min}(\text{minsum}(a[0:m-1]), \text{ms}(m)), 0 \leq m \leq n$$

Because Recurrence 1 contains the function of $\text{ms}(m)$, we need to find the relationship between $\text{ms}(m)$ and $\text{ms}(m-1)$:

$$\text{ms}(m) = (\text{MIN } i: 0 \leq i \leq m: \text{sum}(i, m))$$

$$= (\text{MIN } i: 0 \leq i \leq m: (\sum_{k=i}^{m} a[k]))$$

$= [\text{range division and single range}]$

$$=(\text{MIN } i: 0 \leq i \leq m: (\sum_{k=i}^{m-1} a[k]) + a[m])$$

$= [\text{General distribution law and Definition of sum}]$

$$=(\text{MIN } i: 0 \leq i \leq m: \text{sum}(i, m-1)) + a[m]$$

$= [\text{range division and single range}]$

$$= \text{min}(\text{min}(\text{MIN } i: 0 \leq i \leq m: \text{sum}(i, m-1)), \text{sum}(m, m-1))$$

The second recursive relationship is:

Recurrence 2. Compared with time complexity obtained by combining the Initiation 1, Recurrence 1 and the loop invariant in the fourth step are respectively expressed.

Function method $\text{sum}(a: \langle \text{int} \rangle, i: \text{int}, j: \text{int})$: int

- Write the loop invariant
  We can get the loop invariant by storing $\text{ms}(a[0:m-1])$ and $\text{ms}(m)$ into the variable $s$ and $c$.

  $\rho: 0 \leq m \leq n / \wedge s = \text{min}(\text{ms}(a[0:m-1])) / \wedge c = \text{ms}(m)$

- Generate the Dafny algorithm program
  Then, the algorithm with time complexity of $O(n)$ is obtained by combining the Initiation 1, Recurrence 1 and Recurrence 2. Compared with time complexity $O(n^3)$ of the exhaustive method, the algorithm has better optimization efficiency. Algorithm program is as follows:

  method minsum($a: \langle \text{int} \rangle$) returns($s: \text{int}$)
  {
    var $m, c := 0, 0; \quad s := 0;$
    while($m < a.\text{Length}$)
      decreases $a.\text{Length} - m$
      {
        $c := \text{min}(c + a[m], a[m]);$
        $s := \text{min}(s, c);$
        $m := m + 1;$
      }
  }

  The $Q, R$, auxiliary functions in the first step, and the loop invariant in the fourth step are respectively expressed by Dafny and added to the Dafny program for verification. The Dafny program for the smallest sum of contiguous subarrays is as follows:

  function method $\text{sum}(a: \langle \text{int} \rangle, i: \text{int}, j: \text{int})$: int
    reads $a$
    requires $a! = \text{null}$
    requires $0 \leq i \leq j \leq a.\text{Length}$
    decreases $j - i$
    {
      //
    }
3.2 New Local Bubble Sorting

Problem description: Arrange the given integer array a [0: n-1] in no descending order.

- Provide problem specification
  Using BubbleSort(a,0,n) to represent the post-conditions of Bubble Sort.
  \[ Q: a != \text{null} \land n > 0 \]
  \[ R: \text{BubbleSort}(a,0,n) = (\forall j: 1 \leq j < n: a[j-1] \leq a[j]) \land \text{perm}(a,b,0,n-1) \]

The auxiliary function \( \text{perm}(a,b,0,n-1) \) is defined as follows:
\[
\text{perm}(a,b,0,n-1) = (\forall i: 0 \leq i < n: (Nj: 0 \leq j < n: a[j] = a[i]))
\]
- Partition the original problem
  Dividing the original problem into sub-problems with the same structure, constructing a recursive relationship for solving the problem and a preliminary algorithm.

\[
\text{BubbleSort}(a, m, n) = F(\text{BubbleSort}(a, m+1, n), \text{BubbleStep}(a, m, n)), 0 \leq m \leq n
\]

- Find the recurrence relations
  Dividing the original problem and finding the recursive relationship. Defining partition function through constructing sub-problems.

\[
\text{BubbleSort}(a, m, n) = (\forall j: m \leq j < n: a[j-1] \leq a[j])
\]
- [cross-product nature]
\[
(\forall i, j: m \leq j < n: \forall k: a[k-1] \leq a[j])
\]
- [range division and single range]
\[
\text{BubbleSort}(a, m+1, n) \land (\forall j: m \leq j < n: a[j-1] \leq a[j])
\]

Dafny 2.3.0.10506

Dafny program verifier finished with 4 verified, 0 errors
Running...

minsum: -13

Fig.2 Minsum verification result
two recursive relations.

\( \text{BubbleSort}(a, i, n) = \text{BubbleSort}(a, i+1, n) \land \text{BubbleStep}(a, i, n), \ i:1..n \)

\( \text{BubbleStep}(a, i, j) = \text{BubbleStep}(a, i, j-1) \land a[j-1] \leq a[j] \)

- Write the loop invariant

Combining the definitions of the recursive relation \( \text{BubbleSort} \) and \( \text{BubbleStep} \) and the variable scope of \( i, j \), we can obtain the loop invariant:

\[ p: \ \text{BubbleSort}(a, i, n) = (\forall i:m+1 \leq i < n: (\forall j:m \leq j < n: a[j-1] \leq a[j])) \land \text{BubbleStep}(a, i, j-1) = \land a[j-1] \leq a[j]; \ i:1..n, f, i, j, 1 \]

- Generate the Dafny algorithm program

Here are two Dafny methods to achieve the above recursive relationship:

```dafny
method BubbleSort(a: array (int))
{
    if a.Length > 1
    {
        var i := 1;
        while i < a.Length
        {
            invariant 1 <= i <= a.Length;
            decreases a.Length - i;
            BubbleStep(a, i);
            i := i + 1;
        }
    }
}
```

```dafny
method BubbleStep(a: array (int), i: int)
{
    var j := i;
    while (j > 0 && a[j-1] > a[j])
    {
        a[j-1], a[j] := a[j], a[j-1];
        j := j - 1;
    }
}
```

Add \( Q, R \), auxiliary functions, and loop invariants to Dafny methods for verify:

```dafny
predicate permutation (a: seq (int), b: seq (int))
{
    multiset (a) == multiset (b)
}
predicate ord(a: array (int), lo: int, hi: int)
requires a != null && 0 <= lo <= hi <= a.Length
reads a
{
    For all i, j : lo <= i < j < hi ==> a[i] <= a[j]
}
```

```dafny
predicate sorted (a: array (int))
requires a != null
reads a
{
    ord(a, 0, a.Length)
}
```

```dafny
method bubbleSort(a: array (int))
requires a != null
modifies a
ensures sorted(a)
ensures permutation (a[..], ord (a[..]))
{
    if a.Length > 1
    {
        var i := 1;
        while i < a.Length
        {
            invariant 1 <= i <= a.Length;
            invariant ord(a, 0, i);
            invariant permutation (a[..], ord(a[..]));
            decreases a.Length - i;
            bubbleStep(a, i);
            i := i + 1;
        }
    }
}
```

The following are the main method and verification results of the BubbleSort (Fig.3).
method Main()
{
    var a:array<int> := new int[7] [4, 0, 1, 9, 7, 1, 2];
    print "Before: ", a[0], a[1], a[2], a[3], a[4], a[5], a[6], ",n";
    BubbleSort(a);
    print "After: ", a[0], a[1], a[2], a[3], a[4], a[5], a[6], ",n";
}

Fig.3  BubbleSort verification result

4 Conclusion

In this paper, we propose a system method to deduce and synthesize the Dafny programs. First, the specification of problem is described in strict mathematical language. Then, the derivation process uses program specification transformation technology to perform equivalent transformation. Furthermore, Dafny program is synthesized through the obtained recursive relationship and loop invariants. Finally, the functional correctness of Dafny program is automatically verified by Dafny verifier or online tool. Through this method, we deduce and synthesize Dafny programs for many typical problems such as the cube sum problem, the minimum (or maximum) contiguous subarray problems, several searching problems, several sorting problems. Due to space limitation, we only illustrate the development process of Dafny programs for two typical problems: the minimum contiguous subarray problem and the new local bubble sorting problem. It proves that our method can effectively improve the correctness and reliability of Dafny program developed. What’s more, we demonstrate the potential of the deductive synthesis method by developing a new local bubble sorting program.

Our next plan is to use this method and combine with our previous work\cite{26-31} to deduce and synthesize more complex nonlinear data structure algorithms, such as the binary tree or graph related algorithms.

References


