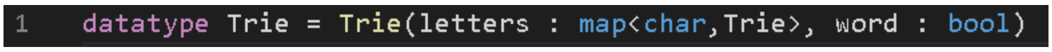
SWEN324 2020 A4 ESSAY

Student Name: Jingjing Chen
Student ID: 300486503

Dafny is a programming language and program verifier. A Dafny program includes specifications, code, inductive invariants, and termination metrics. The program verifier checks program correctness and all proof steps [1].

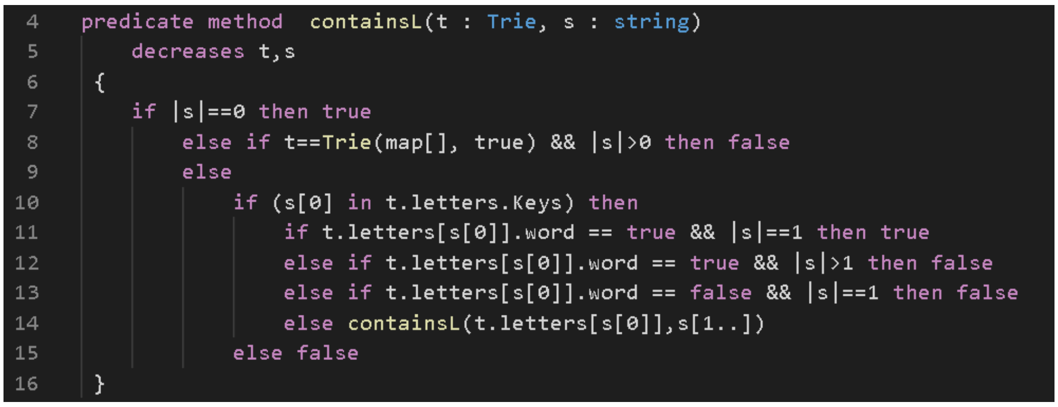
Here we give some Dafny programs used as examples in the essay.

Trie is an effective information retrieval data structure. It is also called prefix tree, which is a multi-tree structure.



```
datatype Trie = Trie(letters : map<char,Trie>, word : bool)
```

Let's start with the inductive data type of the trie tree. At first, we need complete the predicate specification containsL: that is true if and only if Trie contains s.



```
predicate method  containsL(t : Trie, s : string)
    decreases t,s
 {
    if |s|==0 then true
        else if t==Trie(map[], true) && |s|>0 then false
        else
            if (s[0] in t.letters.Keys) then
                if t.letters[s[0]].word == true && |s|==1 then true
                else if t.letters[s[0]].word == true && |s|>1 then false
                else if t.letters[s[0]].word == false && |s|==1 then false
                else containsL(t.letters[s[0]],s[1..])
            else false
 }
```

A boolean function is also known as a predicate and Dafny reserves a keyword for this purpose: the meaning of predicate ContainsL(t: Trie, s: string) is identical to function ContainsL(t: Trie, s: string): bool

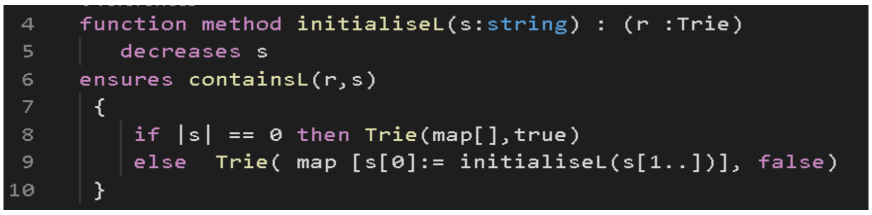
By default, the predicate is ghost. Here, the keyword phrase predicate method means that it can be compiled and invoked.

Destructors are members of the datatype, which means they are accessed by following an expression with a dot and the name of the member [2]. Thus, whereas function method ContainsL is invoked as ContainsL(t, s) for an expression t of type Trie, a destructor is invoked as t.letter[s[0]].

- Tries(map[], true) represents an empty Trie.
- t.letters will return a set of map.
- t.letters [s[0]] is the child Trie corresponding to node s[0].
- Boolean type word is flag, which is set to mark whether the node constitutes a word.

It also checks that programs terminate and are free of evaluation errors, like indexing an array outside its bounds. In this case, If s[0] is not in t.letters.Keys and we still try to create a new node or use a clause containing s[0], Dafny complains that the element may not exist.

Another function method example also shows Dafny is easy to specify, implement and verify our problem. The initialiseL: It returns a Trie containing only the string s.



```
function method initialiseL(s:string) : (r :Trie)
   decreases s
ensures containsL(r,s)
 {
    if |s| == 0 then Trie(map[],true)
    else  Trie( map [s[0]:= initialiseL(s[1..])], false)
 }
```

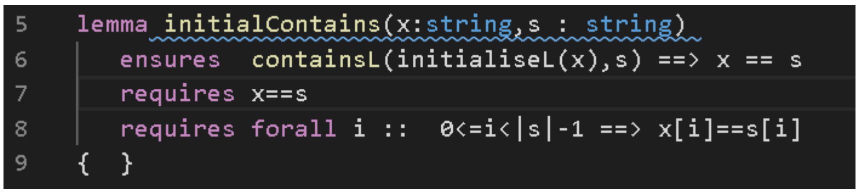
As can be seen in the above code, it is easy to complete the body.

Function method can appear in expressions. And for initialiseL, we write decreases s. These proof obligations imply that initialiseL 's recursion terminates. This recursive call is to avoid Infinite Recursion.

So how do I prove that my specification of function method is correct?

A lemma, also called as a theorem, is a mathematical assertion that has a proof. In Dafny, a lemma is very much like a method: it has a name, it can be parameterized, it has a pre- and postcondition, and it can be called. The logical assertion made by the lemma is declared by the postcondition.

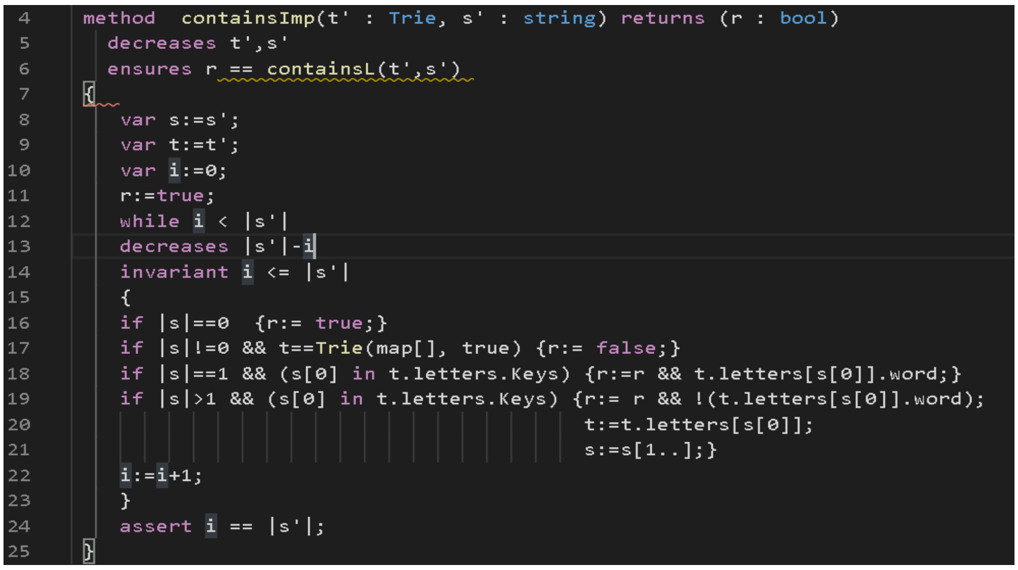
The lemma initialContains : verifying that the Trie built containing the string s only contains the string s



```
lemma initialContains(x:string,s : string)
   ensures  containsL(initialiseL(x),s) ==> x == s
   requires x==s
   requires forall i ::  0<=i<|s|-1 ==> x[i]==s[i]
{ }
```

The lemma is used only by the verifier. Some of the lemmas can be proven automatically, with an empty body { ... } , so we e only need to add some pre-conditions and post-condition clauses.

As we mentioned before, recursion is used in ContainsL fuction method. We expected define an iterative implementation in the method containsImp.

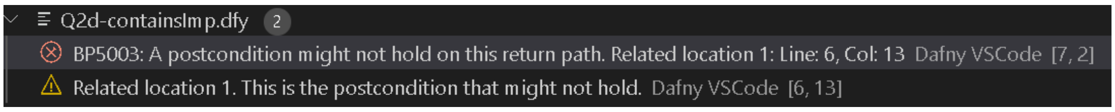


```
method  containsImp(t' : Trie, s' : string) returns (r : bool)
  decreases t',s'
  ensures r == containsL(t',s')
{
   var s:=s';
   var t:=t';
   var i:=0;
   r:=true;
   while i < |s'|
   decreases |s'|-i
   invariant i <= |s'|
   {
   if |s|==0  {r:= true;}
   if |s|!=0 && t==Trie(map[], true) {r:= false;}
   if |s|==1 && (s[0] in t.letters.Keys) {r:=r && t.letters[s[0]].word;}
   if |s|>1 && (s[0] in t.letters.Keys) {r:= r && !(t.letters[s[0]].word);
                                          t:=t.letters[s[0]];
                                          s:=s[1..];}
   i:=i+1;
   }
   assert i == |s'|;
}
```

In the body of the method, we wish to build the required string length iteratively. We will use i to keep track the consecutive number. That is to say, after iterations of the loop, the i equals |s'|.

In Dafny, the loop invariant is identified by the keyword invariant [3]. The loop invariant is an expression that is valid when entering a loop and every execution of the loop body. It captures that each step of the loop is unchanged. That is, no matter where the loop progresses, it will not change. Like pre-conditions and post-conditions, invariants are attributes reserved for each loop execution, expressed in Boolean expressions.

The ensures clause checks that the implementation satisfies the specification containsL that we previously wrote. But Dafny complaints that as follows feedback:



```
Q2d-containsImp.dfy 2
BP5003: A postcondition might not hold on this return path. Related location 1: Line: 6, Col: 13 Dafny VSCode [7, 2]
Related location 1. This is the postcondition that might not hold. Dafny VSCode [6, 13]
```

If the specification we have written is not the right one, we would prove conformance to a wrong requirement. This always happen but sometimes there is no silver bullet to fix this problem.

For instance, even Dafny is executable which enables us to obtain the correct output, and we get some confidence that specifications are the right ones. But there are still some complains which make us confused and would not fix it. In addtion, no standard library is one of Dafny's disadvantages.

In this case Dafny is expected to have some feedback mechanisms to help investigate what steps of a proof cannot be verified and provide more proofs that we can check.

(746 words)

This essay including code can be accessed by:
https://github.com/chenjing999/swen324_Dafny

Reference

[1] K. R. M. Leino, "Developing verified programs with Dafny," *2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, 2013, pp. 1488-1490, doi: 10.1109/ICSE.2013.6606754.

[2] J. Koenig and K. R. M. Leino, “Getting started with Dafny: A guide,” in Software Safety and Security: Tools for Analysis and Verification, ser. NATO Science for Peace and Security Series D: Information and Communication Security. IOS Press, 2012, vol. 33, pp. 152–181.

[3] L. Herbert, K. R. M. Leino, and J. Quaresma, “Using Dafny, an automatic program verifier,” in LASER, International Summer School 2011, ser. LNCS, vol. 7682. Springer, 2012, pp. 156–181.