Assignment 2 - Program Verification Using Dafny

Gaurang Tandon

March 20, 2020

Contents

1 Introduction

In this assignment, you should be able to write simple programs implementing algorithmic systems and verify them using dafny. As a part of the assignment you are expected to

1. Define the transition system for the given problem.
2. Define the pre and post conditions for the initial and terminal states of the transition system.
3. Write the program implementing the transition system in dafny
4. Write the pre and post conditions in the programme as defined in the transition system
5. Run the program and see if your program satisfies the pre and post conditions. The challenge here is to identify as much of pre and post conditions that you can think of.

Refer the following examples and follow the same format for submission.

1.1 Computing max of two numbers

1.1.1 Problem Definition

1. We define the problem as a function $Max : A \rightarrow B$
2. A is the input space defined as $A = \mathbb{Z} \times \mathbb{Z}$
3. B is the output space defined as $B = \mathbb{Z}$

1.1.2 Transition System Definition

$S_{max} =< X, X^o, U, \rightarrow, Y, h >$

1. We define the state space of the system as $X = A \times B$ (cross product of input and output space) , $X = \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$
2. We define an initialization function $\rho : A \rightarrow X$, which converts the input space to state space.
3. $\rho(a, b) = X^o = (a, b, c)$ where $a, b, c \in \mathbb{Z}$ and $c = a$
4. U = {next}
5. Transition relation `def` $(a, b, c) \xrightarrow{next} (a, b, b)$ if b > c $
6. Y = B, As the output space of the system is equal to the output space of the problem.
7. $h : X \rightarrow Y$, as $Y = B \implies h : X \rightarrow B$.
8. $h(x) = x[2]$, where $x \in X$ and $x[2]$ is the 3^{rd} element from the 3 tuple state vector.

1.1.3 Program

```
// Input Space
datatype InputSpace = InputSpace(a:int, b:int)

// State Space
datatype StateSpace = StateSpace(a:int, b:int, c:int)

// rho function
function method rho(tup:InputSpace) :  StateSpace
{
   StateSpace(tup.a,tup.b,tup.a)
}

// view function h
function method pi(trip:StateSpace) : int
{
(trip.c)
}
```

```
// Transition System
method MaxTransitionSystem(initState:StateSpace) returns (terminalState:StateSpace)
//pre-condition
requires initState.a == initState.c
// post-conditions:
ensures terminalState.a < terminalState.b ==> terminalState.c == terminalState.b
ensures terminalState.b <= terminalState.a ==> terminalState.c == terminalState.a
ensures terminalState.c > terminalState.b ==> terminalState.c >= terminalState.a
ensures terminalState.c == terminalState.a ==> terminalState.c >= terminalState.b
ensures terminalState.c == terminalState.b ==> terminalState.c >= terminalState.a
ensures terminalState.c >= terminalState.a && terminalState.c >= terminalState.b
ensures terminalState.a == terminalState.b ==> terminalState.c == terminalState.a
&& terminalState.c == terminalState.b
ensures initState.a > initState.b ==> terminalState.c == initState.a
ensures initState.b > initState.a ==> terminalState.c == initState.b
{
// actual definition begins here
var a := initState.a;
var b := initState.b;
var c := initState.c;
if (a < b) {
c:=b;
}
terminalState := StateSpace(a,b,c);
return terminalState;
}

// Orchestrator
method Main()
{
var inputParameters := InputSpace(3,5);
var initialState := rho(inputParameters);
var terminalState := MaxTransitionSystem(initialState);
var output := pi(terminalState);
// Assertions use the pre/post-conditions to simplify
// We don't care at all what happens inside each method
// when we call it, as long as it satisfies its annotations.
assert output == 5;
```

}

1.1.4 Pre Conditions

Define the pre conditions used.

1.1.5 Post Conditions

- ensure that if a is less than b, implies that c is equal to b

Define the post conditions used.

1.2 Find the factorial of a number

Given a positive number, find it's factorial.

1.2.1 Problem Definition

1. We define the problem as a function $Fact : \mathbb{Z} \to \mathbb{Z}$
2. The input as well as the outspace is $\mathbb{Z}$

1.2.2 Transition System Definition

1. $S_{fact} =< X, X^o, U, \to, Y, h >$
2. The state space of the system $X = \mathbb{Z} \times \mathbb{Z}$
3. We define a function $\rho : \mathbb{Z} \to X$, which converts the input space of the problem to the state space of the system
4. $\rho(n) = (n, 1)$, such that $n \in \mathbb{Z}$ is the case for the initial state. Hence, $X^o = \rho(n) = (n, 1)$.
5. U = {next}
6. Transition Relation $(a, b) \xrightarrow[next]{fact} (a-1, b*a)$, such that $a, b \in \mathbb{Z} \wedge a, b > 0$
7. We define a transition function $t : X \to X$, and t^n as the n^{th} iterate of function t, where $n \in \mathbb{Z} \wedge n > 0$ defined by $t^0 = t, t^1 = t \circ t, t^n = t \circ t....(n-1)times.... \circ t = t \circ t^{n-1}$

8. Let X_f be the final state of the system, defined as $X_f = t^n(a, b)$ iff $a = 0$. Now t^0 corresponds to X^o, and likewise t^n corresponds to X_f. Which means $X^o \xrightarrow{*} X_f = t^n$

9. $Y = \mathbb{Z}$, as the view space of the system is equal to the output space of the problem

10. $h : X \to Y$, where $h : X \to \mathbb{Z}$

1.2.3 Program

```
// State Space
datatype StateSpace  = StateSpace(i:int,a:int)

function fact(i:int): int
decreases i
{
if i >0 then i * fact(i-1) else 1
}

// Transition System
method FactorialTransitions(initialState: StateSpace) returns (finalState: StateSpace)
// pre conditions
requires initialState.i >= 0
requires initialState.a == 1
//post condition
ensures finalState.i == 0
  ensures finalState.a >= 1 ==> initialState.a >= 1
ensures finalState.a == fact(initialState.i)
{
var n := initialState.i;
var i: int := n;
if i == 0 {
return  StateSpace(0,1);
}
else{
var f := 1;
while i >= 1
// loop invariance
decreases i
invariant 0 <= i <= n
```

```
invariant fact(i)*f == fact(n)
{
f := f * i;
i := i - 1;
}
return StateSpace(i,f);
}

}

// Converts state space to output space
function method pi(state: StateSpace): int
{
state.a
}

// Converts input space to state space
function method rho(n:int) : StateSpace
{
StateSpace(n,1)
}

// Orchestrator
method Main(){

var initialState := rho(5);
var terminalState := FactorialTransitions(initialState);
var f := pi(terminalState);
assert f == 120;
}
```

1.2.4 Pre Condition

- the input integer is always less than or equal to -1

```
requires x <= -1
```

1.2.5 Post Conditions

- ensure that every output value should be greater than or equal to zero

```
ensures 0 <= y
```

- ensure that if x is greater than or equal to zero, implies that x will be equal to y

```
ensures 0 <= x ==> x == y
```

- ensure that if x is less than zero, implies that y will be a negation of x i.e. y = -(x)

```
ensures x < 0 ==> y == -x
```

2 Assignment Problems

2.1 Question 1

Write an iterative program which computes the Fibonacci for a given integer. The program should make use of a while / for loop as per the dafny syntax.

2.1.1 Transition System Definition

1. $S_{fib} =< X, X^o, U, \rightarrow, Y, h >$
2. The state space of the system $X = \mathbb{N} \times \mathbb{N} \times \mathbb{N}$
3. We define a function $\rho : \mathbb{N} \rightarrow X$, which converts the input space of the problem to the state space of the system
4. $\rho(n) = (n, 0, 1)$, such that $n \geq 1$ is the case for the initial state. Hence, $X^o = \rho(n) = (n, 0, 1)$.
5. U = {next}
6. Transition Relation:
   - $(n, a, b) \xrightarrow[\text{next}]{\text{fib}} (n-1, b, b+a)$, such that $a, b \in \mathbb{N}$
   - $(1, a, b)$ is terminal state
7. We define a transition function $t : X \rightarrow X$, and t^{n} as the n^{th} iteration of function t, where $n \in \mathbb{Z} \wedge n > 0$ defined by $t^0 = t, t^1 = t \circ t, t^n = t \circ t....(n-1)times.... \circ t = t \circ t^{n-1}$

8. Let X_f be the final state of the system, defined as $X_f = t^n(a, b, c)$ iff $a = 1$. Now t^0 corresponds to X^o, and likewise t^n corresponds to X_f. Which means $X^o \xrightarrow{*} X_f = t^n$

9. $Y = \mathbb{N}$, as the view space of the system is equal to the output space of the problem

10. $h : X \to Y$, where $h : X \to \mathbb{N}$

2.1.2 Fibonacci Iterative Program

```
/**
 * Assuminig
 * F_1 = 0
 * F_2 = 1
*/


// using nat datatype since fibonacci is always positive
datatype StateSpace = StateSpace(n: nat, a: nat, b: nat)

function fib(n: nat) : nat
requires n >= 1
decreases n
{
if (n == 1) then 0 else if (n == 2) then 1 else fib(n - 1) + fib(n - 2)
}

method FibonacciTransitions(initialState: StateSpace) returns (finalState: StateSpace)
requires initialState.n >= 1
requires initialState.a == 0
requires initialState.b == 1
ensures finalState.n == 1
ensures finalState.a == fib(initialState.n)
ensures finalState.b == fib(initialState.n + 1)
{
var n := initialState.n;
var a := initialState.a;
var b := initialState.b;

var i := 1;
```

```
var nOrg := n + 1;

while n > 1
// loop measure
decreases n
invariant n >= 1
// we need to help Dafny get sense of
// the fact that i == nOrg on loop exit
// so that it is sure that a == fib(nOrg)
// on exit
invariant n + i == nOrg;

// necessary loop condition
invariant a == fib(i)
invariant b == fib(i + 1)
{
a, b := b, a + b;
n := n - 1;
i := i + 1;
}

// just sanity checks
assert n == 1;
assert i == nOrg - 1;

finalState := StateSpace(n, a, b);
}


function method pi(state: StateSpace): nat
requires state.n == 1
{
state.a
}

function method rho(n: nat) : StateSpace
{
StateSpace(n, 0, 1)
}
```

```
method Main(){
assert fib(1) == 0;
assert fib(2) == 1;

var n := 10;
var initialState := rho(n);
var finalState := FibonacciTransitions(initialState);
var answer := pi(finalState);

assert answer == 34;
}
```

2.1.3 Pre Conditions

- $n \geq 1$
- $a = 0$
- $b = 1$

2.1.4 Post Conditions

- $n = 1$
- $a = \text{fib}(\text{initial.n})$
- $b = \text{fib}(\text{initial.n} + 1)$

2.2 Question 2

Write a recursive program which computes the Fibonacci for a given integer. The program should **not** make use of a while / for loop. Use appropriate recursive structure.

2.2.1 Transition System Definition

1. $S_{fib} =< X, X^o, U, \rightarrow, Y, h >$
2. The state space of the system $X = \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$
3. We define a function $\rho : \mathbb{N} \rightarrow X$, which converts the input space of the problem to the state space of the system

4. $\rho(n) = (n, 1, 0, 1)$, such that $n \geq 1$ is the case for the initial state. Hence, $X^o = \rho(n) = (n, 1, 0, 1)$.

5. U = {next}

6. Transition Relation:

   - $(n, i, a, b) \xrightarrow[\text{next}]{\text{fib}} (n, i+1, b, b+a)$, such that $a, b \in \mathbb{N}$ and $a = F_i$ and $b = F_{i+1}$
   - (n, n, a, b) is terminal state, where $a = F_n$ and $b = F_{n+1}$

7. We define a transition function $t : X \to X$, and $\mathrm{t^n}$ as the $\mathrm{n^{th}}$ iteration of function t, where $n \in \mathbb{Z} \wedge n > 0$ defined by $t^0 = t, t^1 = t \circ t, t^n = t \circ t....(n-1) times.... \circ t = t \circ t^{n-1}$

8. Let $\mathrm{X_f}$ be the final state of the system, defined as $X_f = t^n(a, b, c, d)$ iff $a = b$, $a = F_n$ and $b = F_{n+1}$. Now $\mathrm{t^0}$ corresponds to $\mathrm{X^o}$, and likewise $\mathrm{t^n}$ corresponds to $\mathrm{X_f}$. Which means $X^o \xrightarrow{*} X_f = t^n$

9. $Y = \mathbb{N}$, as the view space of the system is equal to the output space of the problem

10. $h : X \to Y$, where $h : X \to \mathbb{N}$

2.2.2 Fibonacci Recursive Program

```
// using nat datatype since fibonacci is always positive
datatype StateSpace = StateSpace(n: nat, i: nat, a: nat, b: nat)

function fib(n: nat) : nat
requires n >= 1
decreases n
{
if (n == 1) then 0 else if (n == 2) then 1 else fib(n - 1) + fib(n - 2)
}

method FibonacciTransitions(currState: StateSpace) returns (finalState: StateSpace)
decreases currState.n - currState.i
requires 1 <= currState.i <= currState.n
requires currState.a == fib(currState.i)
requires currState.b == fib(currState.i + 1)
ensures finalState.n == finalState.i
```

```
ensures finalState.a == fib(currState.n)
ensures finalState.b == fib(currState.n + 1)
{
var n, i, a, b := currState.n, currState.i, currState.a, currState.b;

if(i == n) {
finalState := currState;
} else {
var nextState := StateSpace(n, i + 1, b, a + b);
finalState := FibonacciTransitions(nextState);
}
}

function method pi(state: StateSpace): nat
{
state.a
}

function method rho(n: nat) : StateSpace
requires n >= 1
{
StateSpace(n, 1, 0, 1)
}

method Main(){
var n := 10;
var initialState := rho(n);
var finalState := FibonacciTransitions(initialState);
var answer := pi(finalState);

assert answer == 34;
}
```

2.2.3 Pre Conditions

- $n \geq 1$
- $i = 1$
- $a = 0$

- $b = 1$

2.2.4 Post Conditions

- $n = i =$ initial.n
- $a =$ fib(initial.n)
- $b =$ fib(initial.n + 1)

2.3 Question 3

Write a program for bubble sort which takes input as an integer array and produces a sorted array using bubble sort algorithm.

2.3.1 Transition System Definition

1. $S_{fib} =< X, X^o, U, \rightarrow, Y, h >$
2. The state space of the system $X = \mathbb{Z}^{\mathbb{N}} \times \mathbb{N}$ (integer array of natural length)
3. We define a function $\rho : \mathbb{N} \rightarrow X$, which converts the input space of the problem to the state space of the system
4. $\rho(n) = (\text{arr}, n)$, such that $n =$ arr.Length is the case for the initial state. Hence, $X^o = \rho(n) = (\text{arr}, n)$.
5. U = {next}
6. Transition Relation:
   - $(\text{arr}, n) \xrightarrow[\text{next}]{\text{sort-pass}} (\text{arr'}, n-1)$, such that $n \in \mathbb{N}$ and $n \geq 2$ and arr' is such that the last 'arr.Length - n + 1' elements in the arr have been sorted in arr', and also they appear contiguously in the last indices of the array. (Basically, this is one succesful completion of a bubble sort pass)
   - $(\text{arr}, 1)$ is terminal state
7. We define a transition function $t : X \rightarrow X$, and t^{n} as the n^{th} iteration of function t, where $n \in \mathbb{Z} \wedge n > 0$ defined by $t^0 = t, t^1 = t \circ t, t^n = t \circ t....(n-1)times.... \circ t = t \circ t^{n-1}$

8. Let X_f be the final state of the system, defined as $X_f = t^n(\text{arr}, 2)$ such that 'arr' is sorted. Now t^0 corresponds to X^o, and likewise t^n corresponds to X_f. Which means $X^o \xrightarrow{*} X_f = t^n$

9. $Y = \mathbb{Z}^{\mathbb{N}}$, as the view space of the system is equal to the output space of the problem

10. $h : X \to Y$, where $h : X \to \mathbb{Z}^{\mathbb{N}}$

2.3.2 Bubble Sort Program

```
// both start and end are indices in the array
predicate sorted(arr: array<int>, start: int, end: int)
reads arr
requires 0 <= start <= end < arr.Length
{
forall idx :: start <= idx < end ==> arr[idx] <= arr[idx + 1]
}

// all elements below idx must be smaller than those above
// and all elmeents above idx must be sorted themselves
predicate sortedAtIndex(arr: array<int>, idx: int)
reads arr
{
forall x, y :: (0 <= x < idx && idx <= y < arr.Length) ==> arr[x] <= arr[y]
}

datatype StateSpace = StateSpace(arr: array<int>, pass: int)

// since 'arr' was not declared as array?<int>, it will always have
// non-null type
method BubbleSortStateTransition(state: StateSpace) returns (finalState: StateSpace)
requires state.arr.Length >= 1
requires state.pass == state.arr.Length
ensures finalState.pass == 1
ensures finalState.arr.Length == state.arr.Length
ensures sorted(finalState.arr, 0, finalState.arr.Length - 1)
{
var n := state.arr.Length;
var newArr := new int[n];
```

```
var i := 0;
while i < n
invariant i <= newArr.Length
invariant forall j :: 0 <= j < i ==> newArr[j] == state.arr[j]
{
newArr[i] := state.arr[i];
i := i + 1;
}
assert forall i :: 0 <= i < n ==> newArr[i] == state.arr[i];

// bubble sort for n elements array
// requires n-1 passes

// index above which all heaviest elements are already collected and sorted
var sortedAbove := state.pass;

// stops once sortedAbove == 1
// i.e., all elements above and including 1 are sorted
// that implies zero-th element is automatically sorted
while sortedAbove >= 2
invariant 0 <= sortedAbove <= n
invariant sortedAtIndex(newArr, sortedAbove)
invariant sortedAbove < n ==> sorted(newArr, sortedAbove, n - 1)
{
var idx := 0;
var farthestIdx := sortedAbove - 2;

while idx <= farthestIdx
invariant 0 <= idx <= farthestIdx + 1
invariant sortedAbove < n ==> sorted(newArr, sortedAbove, n - 1)
invariant sortedAtIndex(newArr, sortedAbove)
invariant forall x :: 0 <= x <= idx ==> newArr[x] <= newArr[idx]
{
if(newArr[idx] > newArr[idx + 1])
{
newArr[idx], newArr[idx + 1] := newArr[idx + 1], newArr[idx];
}

assert newArr[idx] <= newArr[idx + 1];
idx := idx + 1;
```

```
}

sortedAbove := sortedAbove - 1;

assert sortedAbove < n - 1 ==> newArr[sortedAbove] <= newArr[sortedAbove + 1];
assert forall x :: 0 <= x < sortedAbove ==> newArr[x] <= newArr[sortedAbove];
}

finalState := StateSpace(newArr, 1);
}

function method rho(arr: array<int>) : StateSpace {
StateSpace(arr, arr.Length)
}

function method pi(state: StateSpace) : array<int>
reads state.arr
{
state.arr
}

method Main(){
var arr := new int[5];
arr[0], arr[1], arr[2], arr[3], arr[4] := 4, 2, 3, 1, 6;

var sts := rho(arr);
var sts2 := BubbleSortStateTransition(sts);
var sortedArr := pi(sts2);

var i := 0;
while i < sortedArr.Length {
print sortedArr[i];
i := i + 1;
}
assert sorted(sortedArr, 0, sortedArr.Length - 1);
}
```

2.3.3 Pre Conditions

- arr.Length ≥ 1

- pass = arr.Length

2.3.4 Post Conditions

- pass = 1
- finalState.arr.Length == initialState.arr.Length
- finalState.arr is sorted first to last index

3 Submission Guidelines

3.1 Emacs Based Submission

Emacs is what you all are suggested to use to to solve the assignment. Emacs is a powerful editor which is used world wide in the scientific communities. It is a good tool to know and use going forward.

- Follow this template to record your solutions. Use the emacs in org-mode (Open emacs -> Alt+X -> type org-mode).
- Write the definition of transition system in the section provided below each question "**Transition System Definition**"
- Write your code in the code blocks below each question "**begin src —- end src**"
- Write the details of the pre condition in the section provided below each section "**Pre Conditions**"
- Write the details of the post condition in the section provided below each section "**Post Conditions**"

3.2 Alternate Submission

- You can use alternative methods based on your convenience to solve the assignment(Visual Studio, Text Editors etc.)
- Finally make sure that your solutions are recorded in the format specified above (Copy and paste text/code in the suitable sections)

3.3 Submission Format

- Create a folder names "Assignment2" in your github assignment repository
- Put all the solution artefacts inside the folder
- Commit and push the solution

4 Grading Scheme

- Assignment Marks - 40
- Extra Marks - 10
- Total Scorable Marks - 50

Sr.No	Category	Marks
1	Trasition System Definition	10
2	Program in Dafny	10
3	Pre-Conditions	5
4	Post-Conditions	5
5	Showing pre/post conditions in transition system definition	10
6	Thoughtful and clever pre/post conditions (with appropriate explanation) that you could define within your transition system	Extra 10

5 Resources

You could refer to the following resources to understand the sytax and it's usage.

- Dafny Syntax Cheatsheet `https://docs.google.com/document/d/1kz5_yqzhrEyXII96eCF1YoHZhnb_6dzv-K3u79bMMis/edit?pref=2&pli=1`
- Developing Verified Programs with Dafny `http://leino.science/papers/krml233.pdf`
- Type Systems in Dafny `http://leino.science/papers/krml243.html`
- Dafny Reference Manual `https://github.com/dafny-lang/dafny/blob/master/Docs/DafnyRef/out/DafnyRef.pdf`