3.2 Recursive Fibonacci Program

3.2.1 Problem Definition

1. We define the problem as a function Fibo: $Z \rightarrow Z$
2. Input Space as well as Output Space is Z

3.2.2 Transition System Definition

1. $S_{fibo} = < X, X^0, U, \rightarrow, Y, h>$
2. The state space of the system $X = Z \times Z$
3. We define a function $\rho: Z \rightarrow X$, which converts the input space of the problem to the state space of the system.
4. $\rho(n) = (n, 0)$, such that $n \in Z$ is the case for the initial state. Hence $X^0 = \rho(n) = (n, 0)$.
5. U = {call, add}
6. Transition Relation $(n, 0) \xrightarrow{call(1)} (n - 1, 0)$ for $n > 2$ and $(n, 0) \xrightarrow{call(1)} (1, 1)$ for $n = 2$, such that $n \in Z \wedge n >= 2$.
7. Transition Relation $(n, 0) \xrightarrow{call(2)} (n - 2, 0)$ for $n > 2$ and $(n, 0) \xrightarrow{call(2)} (0, 0)$ for $n = 2$, such that $n \in Z \wedge n >= 2$.
8. Transition Relation $(n - 1, b), (n - 2, c) \xrightarrow{add} (n, b + c)$ such that $n, b, c \in Z \wedge n >= 2 \wedge b, c >= 0$.
9. Y = Z, as the view space of the system is equal to the output space of the problem.
10. $h: X \rightarrow Y$, where $h: X \rightarrow Z$
11. $h(x) = x[1]$, where $x \in X$ and $x[1]$ is the 2^{nd} element from the 2 tuple state vector.

3.2.3 Program

```
// Input Space
datatype InputSpace = InputSpace(n: int)

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

// rho function
function method rho(tup:InputSpace): StateSpace
{
   StateSpace(tup.n, 0)
}

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

```
   (trip.a)
}

// test function
function fibo(n: int): int
requires n >= 0
decreases n
{
   if n == 0 then 0
   else if n == 1 then 1
   else fibo(n - 1) + fibo(n - 2)
}

// Transition System
method TransitionSystem(input: StateSpace) returns (output:StateSpace)
// pre conditions
requires input.n >= 0
requires input.a == 0
// post conditions
ensures output.n == input.n
ensures output.a == fibo(output.n)
decreases input.n
{
   var n := input.n;
   var a := input.a;

   if(n == 0)
   {
       a := 0;
   }
   else if(n == 1)
   {
       a := 1;
   }
   else{
       var fibo1 := StateSpace(n - 1, 0);
       fibo1 := TransitionSystem(fibo1);
```

```
        var fibo2 := StateSpace(n - 2, 0);
        fibo2 := TransitionSystem(fibo2);
        a := fibo1.a + fibo2.a;
    }
    output := StateSpace(n, a);
}

method Main()
{
    var input := InputSpace(4);
    var initState := rho(input);
    var terminalState := TransitionSystem(initState);
    var output := pi(terminalState);
    assert output == 3;
}
```

3.2.4 Pre Condition

1. requires input integer to be always greater than or equal to 0
   requires input.n >= 0
2. requires input.a == 0 initially.

3.2.5 Post Condition

1. ensures that the output is for the given input.
   ensures output.n == input.n.
2. ensures that a which is the final output is the correct n^{th} fibonacci number
   ensures output.a == fibo(output.n).