Z3 Assignments, week 5
Group 2

Maths Challenge **3**
Problem description: 3
Solution explanation: 3

3 Cannibals 3 Missionaries: **8**
Problem Description: 8
Solution explanation: 9
Source code: 10

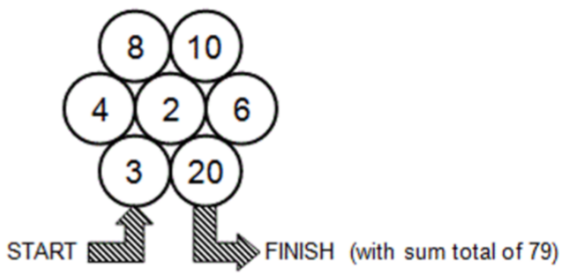
Maths Challenge

Problem description:

Wk 5 Assignments (I)

In the following Maths challenge, you travel from 'START' to 'FINISH' in no specific order or direction from one adjacent circle to another, keeping the accumulating sum total always equal to another 'Prime' number. Values in adjacent circles can be used more than once but 'ping-ponging' between circles (i.e. returning immediately to the circle that you have just left) is not allowed. The following challenge is to finish with a sum total of 79 (a prime number).

Created By Gordon R. Burgin. More at https://gordonburgin.com



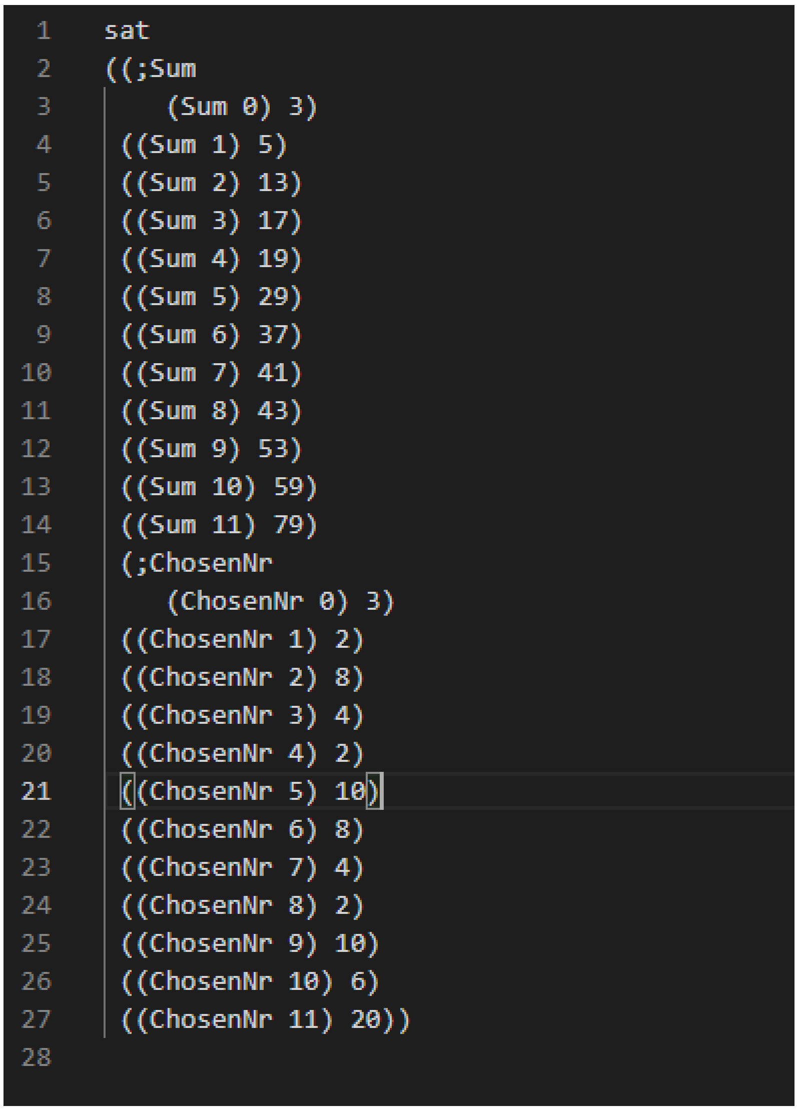
Solution explanation:

In order to approach this problem, we first had to think of a way to specify prime numbers, hence the function IsPrime, that we later let Z3 iterate over all of the sums in a forall quantifier.

In order to let the computer know how to traverse through the 7 “circles” or numbers, a function (AddSum) holds the functionality that specifies where the computer could go based on what circle he is currently on. This is done by having

this function lay in a forall (steps) quantifier, where for each step, depending on the value that it is currently on, can then traverse onto one of its adjacent circles.
Furthermore, we specify that the initial sum should be 3, as well as the initial chosen nr. Similarly, the sum at the end step should be 79 and the traversing should end with the number 20 as the last chosen circle.

In order to avoid ping-ponging between values (going from 3 to 4 and 4 to 3 for example), we have a forall quantifier that specifies that the currently chosen circle/number should be different than the ones 2 moves ahead. Lastly, we make sure that the sum for each step should be a primal number and we finish off by getting the values of all of the chosen numbers for each step and how the sum changes.



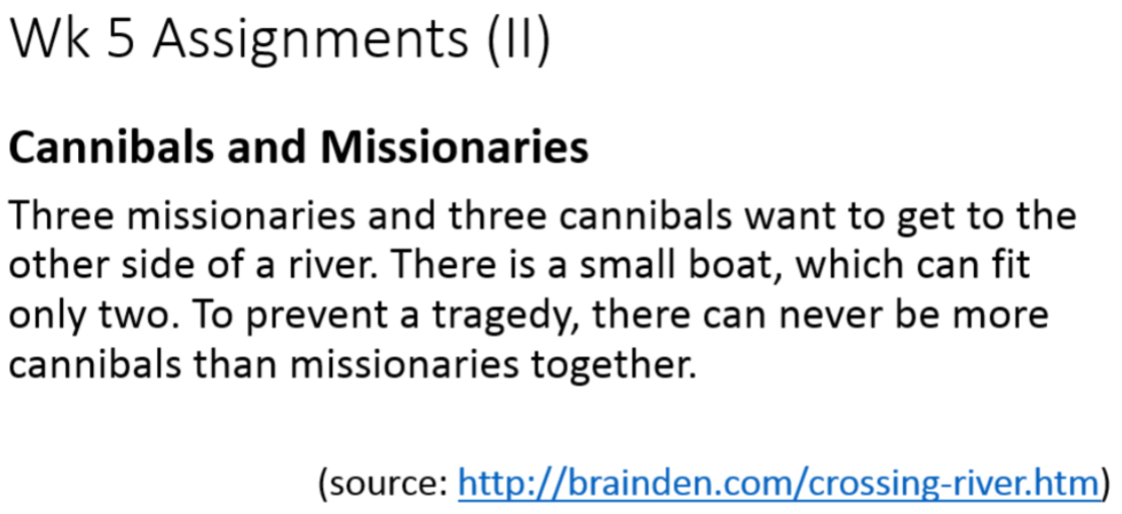
```
sat
((;Sum
    (Sum 0) 3)
 ((Sum 1) 5)
 ((Sum 2) 13)
 ((Sum 3) 17)
 ((Sum 4) 19)
 ((Sum 5) 29)
 ((Sum 6) 37)
 ((Sum 7) 41)
 ((Sum 8) 43)
 ((Sum 9) 53)
 ((Sum 10) 59)
 ((Sum 11) 79)
 (;ChosenNr
    (ChosenNr 0) 3)
 ((ChosenNr 1) 2)
 ((ChosenNr 2) 8)
 ((ChosenNr 3) 4)
 ((ChosenNr 4) 2)
 ((ChosenNr 5) 10)
 ((ChosenNr 6) 8)
 ((ChosenNr 7) 4)
 ((ChosenNr 8) 2)
 ((ChosenNr 9) 10)
 ((ChosenNr 10) 6)
 ((ChosenNr 11) 20))
```

Source code:

```
(declare-fun Sum (Int) Int)
(declare-fun ChosenNr (Int) Int)
(declare-const EndStep Int)
(declare-const S Int)


;A function that allows the check of whether a number is prime or not
(define-fun IsPrime ((value Int)) Bool

    (and
        (> value 0) (not (exists ((y Int) (x Int)) (and (= value (* x
y)) (< x value) (< y value)  (> x 1) (> y 1))))
    )

)

;Add sum holds all of the possible ways Z3 can iterate within the 7
distinct "circles"/numbers
(define-fun AddSum ((step Int)) Bool
    (or
        (and
            (= (ChosenNr step) 3)
            (or
                (= (ChosenNr (+ step 1)) 4) (= (ChosenNr (+ step 1)) 2)
(= (ChosenNr (+ step 1)) 20)
            )
            (= (Sum (+ step 1)) (+ (Sum step) (ChosenNr (+ step 1))))
        )
        (and
            (= (ChosenNr step) 4)
            (or
                (= (ChosenNr (+ step 1)) 8) (= (ChosenNr (+ step 1)) 2)
(= (ChosenNr (+ step 1)) 3)
            )
            (= (Sum (+ step 1)) (+ (Sum step) (ChosenNr (+ step 1))))
        )
        (and
            (= (ChosenNr step) 8)
            (or
                (= (ChosenNr (+ step 1)) 4) (= (ChosenNr (+ step 1)) 2)
(= (ChosenNr (+ step 1)) 10)
            )
            (= (Sum (+ step 1)) (+ (Sum step) (ChosenNr (+ step 1))))
        )
        (and
            (= (ChosenNr step) 10)
            (or
                (= (ChosenNr (+ step 1)) 8) (= (ChosenNr (+ step 1)) 2)
(= (ChosenNr (+ step 1)) 6)
            )
            (= (Sum (+ step 1)) (+ (Sum step) (ChosenNr (+ step 1))))
        )
        (and
            (= (ChosenNr step) 6)
            (or
                (= (ChosenNr (+ step 1)) 10) (= (ChosenNr (+ step 1)) 2)
(= (ChosenNr (+ step 1)) 20)
            )
            (= (Sum (+ step 1)) (+ (Sum step) (ChosenNr (+ step 1))))

        )
        (and
            (= (ChosenNr step) 20)
            (or
                (= (ChosenNr (+ step 1)) 3) (= (ChosenNr (+ step 1)) 2)
(= (ChosenNr (+ step 1)) 6)
            )
            (= (Sum (+ step 1)) (+ (Sum step) (ChosenNr (+ step 1))))
        )
        (and
            (= (ChosenNr step) 2)
            (or
                (= (ChosenNr (+ step 1)) 3) (= (ChosenNr (+ step 1)) 4)
(= (ChosenNr (+ step 1)) 8)
                (= (ChosenNr (+ step 1)) 10) (= (ChosenNr (+ step 1)) 6)
(= (ChosenNr (+ step 1)) 20)
            )
            (= (Sum (+ step 1)) (+ (Sum step) (ChosenNr (+ step 1))))
        )
    )
)


(assert (and

    ;SUM AT STEP 0 is 3
    (= (Sum 0) 3)
    ;NUMBER CHOSEN AT STEP 0 (number is defined by 2)
    (= (ChosenNr 0) 3)

    ;The sum at end step is 79
    (= (Sum EndStep) 79)
    ;The last chosen number is 20
    (= (ChosenNr EndStep) 20)


    ;Ping ponging between values is not allowed
    (forall ((step Int))
        (implies
            (and
                (<= 0 step EndStep)
            )
            (distinct (ChosenNr step) (ChosenNr (+ step 2)))
        )
    )

    (forall ((step Int))
        (implies
            (and
                (<= 0 step EndStep)
            )
            (AddSum step)
        )
    )



    ;FOR ALL STEPS THE SUM SHOULD BE PRIME!!!
    (forall ((step Int))
        (implies
            (and
                (<= 1 step EndStep)
            )
            (IsPrime (Sum step))
        )
    )

    (= EndStep 11)
))

(check-sat)
(get-value (
    ;Sum
    (Sum 0)
    (Sum 1)
    (Sum 2)
    (Sum 3)
    (Sum 4)
    (Sum 5)
    (Sum 6)
    (Sum 7)
    (Sum 8)
    (Sum 9)
    (Sum 10)
    (Sum 11)


    ;ChosenNr
    (ChosenNr 0)
    (ChosenNr 1)
    (ChosenNr 2)
    (ChosenNr 3)
    (ChosenNr 4)
    (ChosenNr 5)
    (ChosenNr 6)
    (ChosenNr 7)
    (ChosenNr 8)
    (ChosenNr 9)
    (ChosenNr 10)
    (ChosenNr 11)


))
```
——---

3 Cannibals 3 Missionaries:

Problem Description:



Wk 5 Assignments (II)

Cannibals and Missionaries

Three missionaries and three cannibals want to get to the other side of a river. There is a small boat, which can fit only two. To prevent a tragedy, there can never be more cannibals than missionaries together.

(source: http://brainden.com/crossing-river.htm)

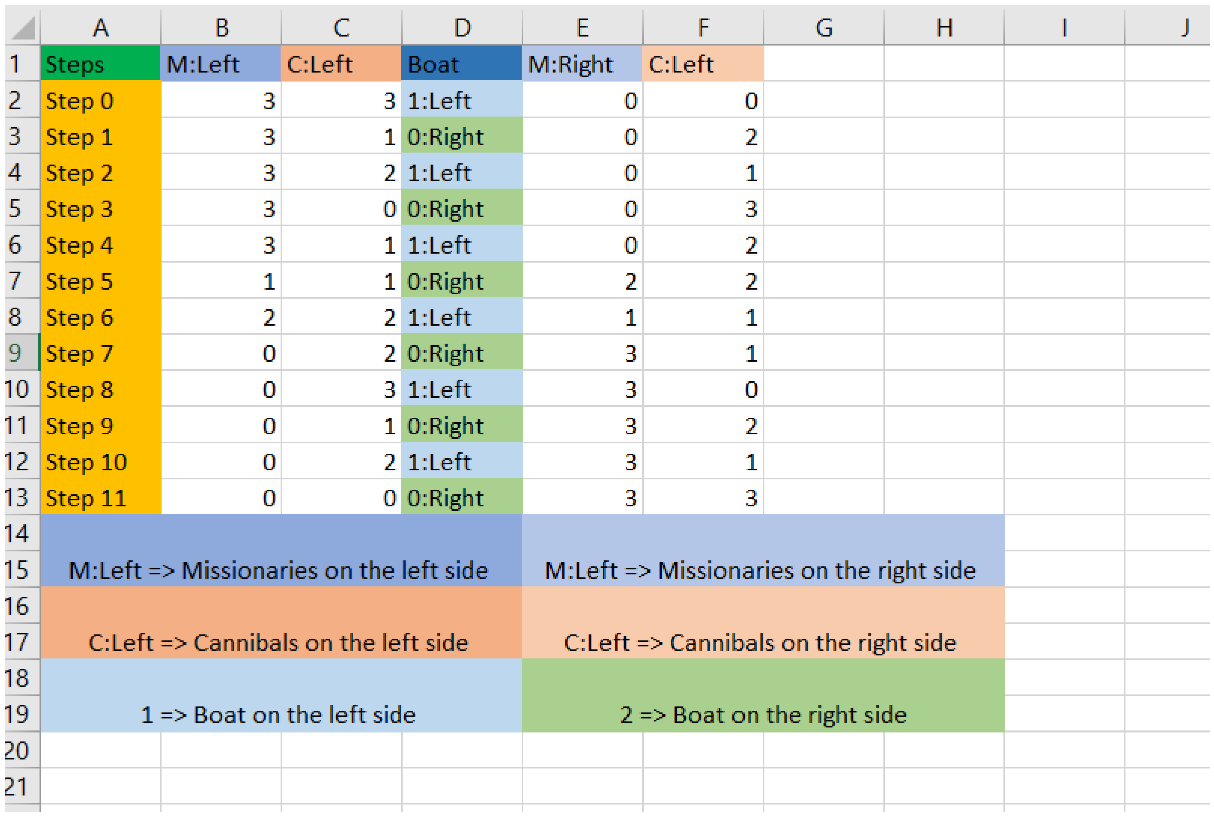
Solution explanation:

In order to solve this puzzle, we had to think about what are the options to move the missionaries and cannibals to the right shore, making sure that the number of cannibals does not exceed the number of missionaries.
Therefore, 2 functions, “Missionaries” and “Cannibals” are declared in order to keep track of the position (which shore the cannibals or the missionaries are) and the timestep, returning the amount of cannibals or missionaries for the respective side.
The constant “EndStep” is used to tell Z3 how many steps it should take to solve the puzzle.

To move the cannibals and the missionaries between the two shores, the function “move” is declared. Within it, we tell Z3 that it has 5 possible options of transporting the persons ({2 cannibals}, {2 missionaries}, {1 cannibal, 1 missionary}, {1 cannibal, 0 missionaries} or {1 missionary, 0 cannibals}).
By using the “total” function, we specify Z3 that the total number of cannibals and missionaries, no matter which shore it is, should be the same.

In the assert part, it is made sure that the initial state and the wanted end-state are mentioned. The first “forall” quantifier specifies that the values of the cannibals and missionaries should be between 1 and 3, while in the second “forall” we tell Z3 that, each time a cannibal or missionary is moved, either the number of cannibals should not exceed the number of missionaries no matter each side they are on and that the number of missionaries should be 0 on the opposite side (so that they are not eaten by cannibals) or that the number of the missionaries and cannibals should be equal on both sides.

The output is as follows:



Steps	M:Left	C:Left	Boat	M:Right	C:Left
Step 0	3	3	1:Left	0	0
Step 1	3	1	0:Right	0	2
Step 2	3	2	1:Left	0	1
Step 3	3	0	0:Right	0	3
Step 4	3	1	1:Left	0	2
Step 5	1	1	0:Right	2	2
Step 6	2	2	1:Left	1	1
Step 7	0	2	0:Right	3	1
Step 8	0	3	1:Left	3	0
Step 9	0	1	0:Right	3	2
Step 10	0	2	1:Left	3	1
Step 11	0	0	0:Right	3	3

M:Left => Missionaries on the left side	M:Left => Missionaries on the right side
C:Left => Cannibals on the left side	C:Left => Cannibals on the right side
1 => Boat on the left side	2 => Boat on the right side

Source code:

```
(declare-fun Missionaries (Int Int) Int)
(declare-fun Cannibals (Int Int) Int)

; boat (step) => Location of boat at given step
;1 = Left beach 'origin', 0 = Right beach 'destination'
(declare-fun boat (Int) Int)
(declare-const EndStep Int)



(define-fun move ((step Int)) Bool
    (ite (= (boat step) 1)
        ; Then case (If boat is on the left beach 'origin')
        (and
            ;pick one of these possible actions
            (xor
                ;two Cannibals
                (and
                    (= (Missionaries 2 (+ step 1)) (Missionaries 2
step))
                    (= (Missionaries 1 (+ step 1)) (Missionaries 1
step))

                    (= (Cannibals 1 (+ step 1)) (- (Cannibals 1 step)
2))
                    (= (Cannibals 2 (+ step 1)) (+ (Cannibals 2 step)
2))


                    (= (boat (+ step 1)) 0)
                )
                ; two Missionaries
                (and
                    (= (Cannibals 2 (+ step 1)) (Cannibals 2 step))
                    (= (Cannibals 1 (+ step 1)) (Cannibals 1 step))

                    (= (Missionaries 1 (+ step 1)) (- (Missionaries 1
step) 2))
                    (= (Missionaries 2 (+ step 1)) (+ (Missionaries 2
step) 2))


                    (= (boat (+ step 1)) 0)
                )
                ; one Cannibal
                (and
                    (= (Missionaries 2 (+ step 1)) (Missionaries 2
step))
                    (= (Missionaries 1 (+ step 1)) (Missionaries 1
step))

                    (= (Cannibals 1 (+ step 1)) (- (Cannibals 1 step)
1))
                    (= (Cannibals 2 (+ step 1)) (+ (Cannibals 2 step)
1))


                    (= (boat (+ step 1)) 0)
                )
                ; one Missionarie
                (and
                    (= (Cannibals 2 (+ step 1)) (Cannibals 2 step))
                    (= (Cannibals 1 (+ step 1)) (Cannibals 1 step))

                    (= (Missionaries 1 (+ step 1)) (- (Missionaries 1
step) 1))
                    (= (Missionaries 2 (+ step 1)) (+ (Missionaries 2
step) 1))


                    (= (boat (+ step 1)) 0)
                )
                ;one Cannibal and one Missionarie
                (and
                    (= (Missionaries 1 (+ step 1)) (- (Missionaries 1
step) 1))
                    (= (Missionaries 2 (+ step 1)) (+ (Missionaries 2
step) 1))

                    (= (Cannibals 1 (+ step 1)) (- (Cannibals 1 step)
1))
                    (= (Cannibals 2 (+ step 1)) (+ (Cannibals 2 step)
1))


                    (= (boat (+ step 1)) 0)
                )
            )
```

```
        )
        ; Else case (If boat is on the left beach 'destination')
        (and
          ;pick one of these possible actions
            (xor
                ;two Cannibals
                (and
                    (= (Missionaries 2 (+ step 1)) (Missionaries 2
step))
                    (= (Missionaries 1 (+ step 1)) (Missionaries 1
step))

                    (= (Cannibals 2 (+ step 1)) (- (Cannibals 2 step)
2))
                    (= (Cannibals 1 (+ step 1)) (+ (Cannibals 1 step)
2))


                    (= (boat (+ step 1)) 1)
                )
                ; two Missionaries
                (and
                    (= (Cannibals 2 (+ step 1)) (Cannibals 2 step))
                    (= (Cannibals 1 (+ step 1)) (Cannibals 1 step))

                    (= (Missionaries 2 (+ step 1)) (- (Missionaries 2
step) 2))
                    (= (Missionaries 1 (+ step 1)) (+ (Missionaries 1
step) 2))



                    (= (boat (+ step 1)) 1)
                )
                ; one Cannibal
                (and
                    (= (Missionaries 2 (+ step 1)) (Missionaries 2
step))
                    (= (Missionaries 1 (+ step 1)) (Missionaries 1
step))

                    (= (Cannibals 2 (+ step 1)) (- (Cannibals 2 step)
1))
                    (= (Cannibals 1 (+ step 1)) (+ (Cannibals 1 step)
1))
```

```
                    (= (boat (+ step 1)) 1)
                )
                ; one Missionarie
                (and
                    (= (Cannibals 2 (+ step 1)) (Cannibals 2 step))
                    (= (Cannibals 1 (+ step 1)) (Cannibals 1 step))

                    (= (Missionaries 2 (+ step 1)) (- (Missionaries 2
step) 1))
                    (= (Missionaries 1 (+ step 1)) (+ (Missionaries 1
step) 1))



                    (= (boat (+ step 1)) 1)
                )
                ;one Cannibal and one Missionarie
                (and
                    (= (Missionaries 2 (+ step 1)) (- (Missionaries 2
step) 1))
                    (= (Missionaries 1 (+ step 1)) (+ (Missionaries 1
step) 1))

                    (= (Cannibals 2 (+ step 1)) (- (Cannibals 2 step)
1))
                    (= (Cannibals 1 (+ step 1)) (+ (Cannibals 1 step)
1))


                    (= (boat (+ step 1)) 1)
                )
            )
        )
    )
)

;Total missionaries and cannibals values are constant
(define-fun total ((step Int)) Int
    (+
        (Missionaries 1 step)
        (Missionaries 2 step)
        (Cannibals 1 step)
        (Cannibals 2 step)
    )
)
```

```
(assert(and
        ;initial state
        (= (Cannibals 1 0) 3)
        (= (Missionaries 1 0) 3)
        (= (Cannibals 2 0) 0)
        (= (Missionaries 2 0) 0)
        (= (boat 0) 1)

         ; the values of each side
         (forall((step Int))
            (=>
                (and
                  (<= 0 step EndStep)
                )
                (and

                    (<= 0 (Missionaries 1 step) 3)
                    (<= 0 (Missionaries 2 step) 3)
                    (<= 0 (Cannibals 1 step) 3)
                    (<= 0 (Cannibals 2 step) 3)
                    (<= 0 (boat step) 1)
                    (= (total step) (total (+ step 1))) ;Total values
combined remain constant
                )
            )
         )


        (forall ((step Int))
          (=>
            (and
                (<= 0 step EndStep)
            )
            (and
                ; pick on of these possible actions:
                (xor
                    (and
                        (or
                            ; If the missionaries are bigger than the
cannibals on the right side, then the missionaries on the left side
should be 0
                            (and
                                (< (Cannibals 2 (+ step 1))
(Missionaries 2 (+ step 1)))
                                (= (Missionaries 1 (+ step 1)) 0)
```

```
                             )
                             ; same logic applies here
                             (and
                                 (< (Cannibals 1 (+ step 1))
(Missionaries 1 (+ step 1)))
                                 (= (Missionaries 2 (+ step 1)) 0)
                             )
                         )
                     )
                     (and
                         (= (Cannibals 1 (+ step 1)) (Missionaries 1 (+
step 1)))
                         (= (Cannibals 2 (+ step 1)) (Missionaries 2 (+
step 1)))
                     )
                 )
                (move step)
            )
          )
        )

        ; final result
        (= (Cannibals 1 EndStep) 0)
        (= (Missionaries 1 EndStep) 0)
        (= (Cannibals 2 EndStep) 3)
        (= (Missionaries 2 EndStep) 3)

        ; How many steps are needed to solve it
        (<= 0 EndStep 11)
    )


)


(check-sat)
(get-value(
    ;step 0
   (Missionaries 1 0)
   (Cannibals 1 0)
   (Missionaries 2 0)
   (Cannibals 2 0)
   (boat 0)
   ;step1
   (Missionaries 1 1)
   (Cannibals 1 1)
```

```
    (Missionaries 2 1)
    (Cannibals 2 1)
    (boat 1)
    ;step 2
    (Missionaries 1 2)
    (Cannibals 1 2)
    (Missionaries 2 2)
    (Cannibals 2 2)
    (boat 2)
    ;step 3
    (Missionaries 1 3)
    (Cannibals 1 3)
    (Missionaries 2 3)
    (Cannibals 2 3)
    (boat 3)
    ;step 4
    (Missionaries 1 4)
    (Cannibals 1 4)
    (Missionaries 2 4)
    (Cannibals 2 4)
    (boat 4)
    ;step 5
    (Missionaries 1 5)
    (Cannibals 1 5)
    (Missionaries 2 5)
    (Cannibals 2 5)
    (boat 5)
    ;step 6
    (Missionaries 1 6)
    (Cannibals 1 6)
    (Missionaries 2 6)
    (Cannibals 2 6)
    (boat 6)
    ;step 7
    (Missionaries 1 7)
    (Cannibals 1 7)
    (Missionaries 2 7)
    (Cannibals 2 7)
    (boat 7)
    ;step 8
    (Missionaries 1 8)
    (Cannibals 1 8)
    (Missionaries 2 8)
    (Cannibals 2 8)
    (boat 8)
    ;step 9
    (Missionaries 1 9)
    (Cannibals 1 9)
    (Missionaries 2 9)
    (Cannibals 2 9)
    (boat 9)
    ;step 10
    (Missionaries 1 10)
    (Cannibals 1 10)
    (Missionaries 2 10)
    (Cannibals 2 10)
    (boat 10)
    ;step 11
    (Missionaries 1 11)
    (Cannibals 1 11)
    (Missionaries 2 11)
    (Cannibals 2 11)
    (boat 11)
    ;step 12
    (Missionaries 1 12)
    (Cannibals 1 12)
    (Missionaries 2 12)
    (Cannibals 2 12)
    (boat 12)

))
```

Output:

```
sat
((;step 0
    (Missionaries 1 0) 3)
  ((Cannibals 1 0) 3)
  ((Missionaries 2 0) 0)
  ((Cannibals 2 0) 0)
  ((boat 0) 1)
  (;step1
    (Missionaries 1 1) 3)
  ((Cannibals 1 1) 1)
  ((Missionaries 2 1) 0)
  ((Cannibals 2 1) 2)
  ((boat 1) 0)
  (;step 2
    (Missionaries 1 2) 3)
  ((Cannibals 1 2) 2)
  ((Missionaries 2 2) 0)
```

```
((Cannibals 2 2) 1)
((boat 2) 1)
(;step 3
  (Missionaries 1 3) 3)
((Cannibals 1 3) 0)
((Missionaries 2 3) 0)
((Cannibals 2 3) 3)
((boat 3) 0)
(;step 4
  (Missionaries 1 4) 3)
((Cannibals 1 4) 1)
((Missionaries 2 4) 0)
((Cannibals 2 4) 2)
((boat 4) 1)
(;step 5
  (Missionaries 1 5) 1)
((Cannibals 1 5) 1)
((Missionaries 2 5) 2)
((Cannibals 2 5) 2)
((boat 5) 0)
(;step 6
  (Missionaries 1 6) 2)
((Cannibals 1 6) 2)
((Missionaries 2 6) 1)
((Cannibals 2 6) 1)
((boat 6) 1)
(;step 7
  (Missionaries 1 7) 0)
((Cannibals 1 7) 2)
((Missionaries 2 7) 3)
((Cannibals 2 7) 1)
((boat 7) 0)
(;step 8
  (Missionaries 1 8) 0)
((Cannibals 1 8) 3)
((Missionaries 2 8) 3)
((Cannibals 2 8) 0)
((boat 8) 1)
(;step 9
  (Missionaries 1 9) 0)
((Cannibals 1 9) 1)
((Missionaries 2 9) 3)
((Cannibals 2 9) 2)
((boat 9) 0)
(;step 10
  (Missionaries 1 10) 0)
```

```
((Cannibals 1 10) 2)
((Missionaries 2 10) 3)
((Cannibals 2 10) 1)
((boat 10) 1)
(;step 11
  (Missionaries 1 11) 0)
((Cannibals 1 11) 0)
((Missionaries 2 11) 3)
((Cannibals 2 11) 3)
((boat 11) 0)
```