Project 1 - Buzdugan Alexandu Msd 2

The pdf can be found in docs/ProjectDescription.pdf

Exercise 1

1. Specify the problem as precondition and postcondition:
   `The precondition is that n has to be an integer and > 1`

   `The postcondition is that m integer, m > n, m % i == 0 && n % i ==0`

2. The program implementation in c is in src and the cink implementation in maude in maude/Exercise1.maude

```
in pl-builtins.maude
in cink-syntax.maude
in tableaux.maude

mod Exercise1 is including CINK-SYNTAX .

ops primality : -> DeclId .
ops exercise1 : -> Stmt .

eq exercise1 =
    int primality ( int n )
    {
        if ( n > 1 ) {
            int m ;
            m = n + 1 ;
            int i ;
            i = 2 ;

            while (m % i != 0 || n % i != 0) {
                if (i > m / 2) { m = m + 1 ; i = 2 ; } else { i = i +
1 ; }
            }
            return m ;
        }
        return -1 ;
    }

    void main () {
    int m ;
    m = primality ( 5 );
    printf("%d;", m) ;
    }
.

endm
```

To execute it run: `maude Exercise1.maude`

3. The tableaux code for our program is in tableaux.maude.

```
red 2pt((n = 5 ; m = n + 1 ; i = 2 ; while (m % i != 0 || n % i != 0)
{ if ( i > m / 2 ){ m = m + 1 ; i = 2 ; } else { i = i + 1 ; }}), (n
<= m && n % i == 0 && m % i == 0)) .

red 2pt((n = 5 ; m = n + 1 ; i = 2 ; while (m % i != 0 || n % i != 0)
{ inv( i > m / 2 ); m = m + 1 ; i = 2 ; }), (n <= m && n % i == 0 && m
% i == 0)) .
```

To execute it run: `maude Exercise.maude`

4. Use v3 to verify the program

For using z3 follow https://rise4fun.com/z3/tutorial

```
The implications are :
    1. i > m / 2[2 / i][m + 1 / m]
    2. n <= m && (n % i == 0 && m % i == 0)
```

Exercise 2

1. We added Exercise2.maude which is a module that contains 3 examples of problems and operations with arrays.

In cink-syntax.maude we added the following operations:

```
- op int_[_]  :  Exp Id -> DeclId [prec 40] .

- op int_[]  :  Exp -> DeclId [prec 40] .

- op _[_]=_ : Exp Id Exp -> Exp [ prec 40 ] .

- _=_[_] : Exp Exp Id -> Exp [ prec 40 ] .

- op _[_]=_[_] : Exp Id Exp Id -> Exp [ prec 40 ] .

- op _[_]=_[_] : Exp Stmt Exp Stmt -> Exp [ prec 40 ] .

- op printf("%d;",_[_]) :  Exp Id -> Exp .
```

2. In Exercise2 we added 3 examples of operations with arrays.

```
in pl-builtins.maude
in cink-syntax.maude

mod Exercise2 is including CINK-SYNTAX .

    ops example1 : -> Stmt .
    ops example2 : -> Stmt .
    ops example3 : -> Stmt .

    eq example1 =

        int a[5];
        int i ;
        i = 2 ;
        a[1] = 7 ;
        a[i * 2] = 5 + a[i + 1] ;
        .

    eq example2 =

        int a[5];
        int b[3] ;
        b = a ;
        printf("%d;", b[1]) ;
        .

    eq example3 =

        int i ;
        i = 0 ;
        int a[5] ;

        while(i < 5){
            a[i] = i + 1 ;
            i = i + 1 ;
            printf("%d;", a[i]) ;
        }

        int b[] = a[5] ;

        i = 0 ;
        while(i < 5){
            b[i] = i * 2 + a[i] ;

            i = i + 1 ;

            a[i] = b[i - 1] ;
        }
        .

endm
```

Exercise 3

1. None

2. Exercise 3 solution implemented with the cink changes to support arrays

```
in pl-builtins.maude
in cink-syntax.maude

mod Exercise3 is including CINK-SYNTAX .
    ops contains0 : -> DeclId .
    ops example1 : -> Stmt .

    eq example1 =
        int contains0 ( int a[], int j ){
            int i = 0 ;

            while(i < j){
                if(a[i] == 0){
                    printf("%d;", i) ;
                    return 0 ;
                }

                i = i + 1 ;
            }

            return -1 ;
        }

        void main(){
            int a[4] = {1, 2, 6, 4, 0} ;
            int i ;
            i = contains0(a, 4) ;

            printf("%d;", i) ;
        }
        .
endm
```

The changes to support this were the addition of dynamic allocation operation:

```
op _[_]={_} : Exp Id List{Exp} -> Exp [prec 0 ] .
```