CSC410

Assignment 6

Due on Sunday December 8, 2020 at 11:59pm

Assignment Format and Guidelines on Submission

This assignment is worth 10% of the total course mark. Submit on Markus and follow these rules:

- This is a group assignment, designed for groups of 4 students. The volume of work is designed so that it would be reasonable for a group of 4. You may choose to work in a group of 3. Groups of fewer than 3 students will be not be accepted. You cannot submit individually.
- Each group should submit two files `q1.rkt` and `q2.rkt`. All the necessary helper functions should be included in each file for each problem. You should not change any of the names or signatures of the functions already provided in the files. You are not allowed to import any new module in the files.

Note that your assignment will be automatically graded. Your functions will be called from another script. If you mess with the signature, the call will fail and the autograder will give you a 0 mark.

The submission system will remain open for 12 hours after the deadline, but there is a penalty deduction formula set in Markus that deducts 4% for every hour of late submission up to 12 hours.

Instructions about Rosette

- For this assignment, you should have Racket (version ≥ 7.0) and Rosette installed on your computer. You have `racket` and `raco` installed on the CDF/TeachingLabs, and you can install Rosette locally using the command:

  ```
  $ raco pkg install rosette
  ```

  The installation instructions are also in the Rosette guide.
- The Rosette guide `https://docs.racket-lang.org/rosette-guide/index.html` contains all the information you need on bitvectors and synthesis constructs. Also, having followed the tutorial on Rosette will help.

Word of Advice

Carefully read the comments in the starter code files before starting your assignment. It will help you understand what you are required to do, as well a give you hints to help you.

Problem 1 (50 points)

The *next larger power of two* of an integer n is the smallest power of two that is larger than n:

$$\forall n > 0, \quad p \text{ is the next larger power of two of } n \iff (n \leq p < 2n) \wedge p \text{ is a power of } 2$$

A naive way of finding the next larger power of two would be to compute every power of two in a loop until it we reach the first that is greater than n. However, it can efficiently be computed for an integer represented on a machine using only bit operations in constant time.

For example, this can be done using only 10 bit-manipulation operations for integers represented on 16 bits:

```
unsigned int n; // Integer represented on 16 bits
n--;
n |= n >> 1;   // Divide by 2^k for consecutive doublings of k up to 16,
n |= n >> 2;   // and then or the results.
n |= n >> 4;
n |= n >> 8;
n++;
```

The goal of this problem is for you to write the procedure that synthesizes this efficient implementation of the next larger power of two for *any integer representation size* up to 64 bits. The number of operations needed will vary, but you can assume that the shape of the solution will stay the same. Since we know the shape of the solution, but not the exact ingredients, syntax guided synthesis is particularly useful to solve this problem.

Solving this problem would involve you providing three key ingredients to the (synthesis) solver: the correctness specification of the problem, a sketch, and the syntax rules defining how to complete the sketch. The correctness specification is a formula that defines the function we are looking for. The sketch is a partial program with holes that can be completed using the syntax rules, and it defines the state space in which the solver will search for a solution.

You are provided with a file `q1.rkt`, in which all the functions that we will list below appear either commented out or with an empty body[1]. Your task is to modify this file. Carefully read the code provided before starting to implement your solutions! The program synthesized by your solution should be similar to the example above. Other types of solutions are not relevant.

The following list explains what functions should be written by you for each of the components of the syntax guided synthesis problem, with constraints on how you should write them.

Correctness specification You have to specify the problem by completing two functions:

- `specification`: Write the body of the function `specification`. It has to take two inputs, a function f and an argument x such that `specification`$(f, x) \iff f(x)$ is the next larger power of two of x. For that, you can write a function `isPowerOfTwo` : *int* → *bool* that checks if its argument is a power of two, preferably using bitvector operations.
- `constraint`: In the bitvector representation, the range of integers that can be represented is limited, so you should not try to synthesize a function that satisfies the specification for any integer, but rather for integers that can be represented by a bitvector of type `bvrepr?`. What is that range? Be careful with the fact that the representation also contains unsigned integers. Encode this restriction in a function `constraint`: *int* → *bool*.

The synthesized solution f will satisfy:

$$\forall x \in \mathbb{N},\ \texttt{constraint}(x) \implies \texttt{specification}(f, x)$$

That is, f will be synthesized such that `specification`(f, x) is true for any integer x that satisfies `constraint`(x).

Sketch To specify the search space for the solution, you need to write a sketch of the solution and provide syntax rules that define how the sketch can be completed. The sketch is a function `next-power` using the syntax `Body?` in its definition.

- Write a first syntax rule `BitVector?` such that a hole `(BitVector?)` will be filled with any bitvector of type `bvrepr?`. Your rule is used in the definition of the two helpers given to you in the code.
- Write another syntax rule `Body?` that describes the grammar generating a sequence of bitwise or/shift operations. **No predefined constant or predefined choice of operator** can appear in the rule. You can use the syntax rules `BVLogic?` and `BVArith?` that are given to you as helpers.

[1] A global parameter `repr_size` is declared at the beginning of the file; it is the current representation size. The bitvectors of size `repr_size` are of type `bvrepr?`.

- Write the complete sketch of your function in `next-power`. The body can use only:
  - the input variable and the bitvector type `bvrepr?`,
  - the rules previously defined (you might have to use an integer constant to set the inlining bound or depth of the rule),
  - the programming construct `let`,
  - and conversion operators such as `integer->bitvector` and `bitvector->integer`.

Solve Once all the components have been provided, the solver can find a solution f that satisfies the specification in the space described by the sketch and the syntax rules. You don't have to write anything here, the `synthesize` construct is already provided in the file `q1.rkt`:

```
(define-symbolic x integer?)
(define solution
  (synthesize
   #:forall (list x)
   #:assume (assert (constraint x))
   #:guarantee (assert (specification next-power x))))
```

Once you have completed the file, executing `racket q1.rkt` should print out a solution for the representation size that is defined at the beginning in `repr_size`. Performance is not an issue in this assignment. Your program should return an answer for any representation size up to 64. You have to use the names we provide for each function and each rule, but you can add your own if you need to define other functions/rules.

Deliverables

Submit the `q1.rkt` file. You shouldn't have changed the name of the functions, so we can automatically test your file and verify that you respected the constraints on what you can put in the different rules and functions. We will test your program with `repr_size` varying between 8 and 64.

Hints and remarks

- You can check that a positive number is a power of two in C using bitwise logical operations:

  ```
  x & (x - 1) == 0 // true if x is a power of two
  ```

- In Racket/Rosette, you need to explicitly convert the integers to bitvectors to use the bitwise operators. The implementation in C seen above can be written in a very straightforward manner:

  ```
  (define (next-power i)
    (let ([v (integer->bitvector i 16)])
      (let ([v (bvsub v (bv 1 16))])                   ;; n--
        (let ([v (bvor v (bvashr v (bv 1 16)))])       ;; n = n | n >> 1
          (let ([v (bvor v (bvashr v (bv 2 16)))])     ;; n = n | n >> 2
            (let ([v (bvor v (bvashr v (bv 4 16))]))   ;; n = n | n >> 4
              (let ([v (bvor v (bvashr v (bv 8 16)))]) ;; n = n | n >> 8
                (let ([v (bvadd v (bv 1 16))])         ;; n ++
                  (bitvector->integer v)))))))))
  ```

- Think about you can use the integer hole `??` to construct a bitvector constant. Simply writing `(bv (??))` will not work.

Problem 2 (50 points)

The goal of this problem is to synthesize a function from input/output examples. The function that needs to be synthesized takes three integer arguments and returns one integer. We do not know the formal specification of the function, but we have the following set of input-output examples (written as $(input \to output)$).

$(2, -4, -1) \to -4$	$(5, 3, -10) \to 3$	$(9, -3, 4) \to 2$
$(12, -4, 2) \to 8$	$(1, -4, 7) \to -2$	$(0, 4, 1) \to 4$
$(-3, 3, 10) \to 8$	$(-10, 9, 3) \to 9$	$(13, 9, 0) \to 10$

A solution is a function that satisfies, for each input/output pair, $f(input) = output$. You are provided with a file `q2.rkt`. Your task is to write a sketch, a specification and an expression grammar in `q2.rkt` such that the code provided at the end of the file prints the solution. The function should be defined using only `+, -, min` and `max` operators, integer constants and as many variables as you need.