CS 395T: Spring 2021
Homework 1: Symbolic Program Synthesis with Rosette

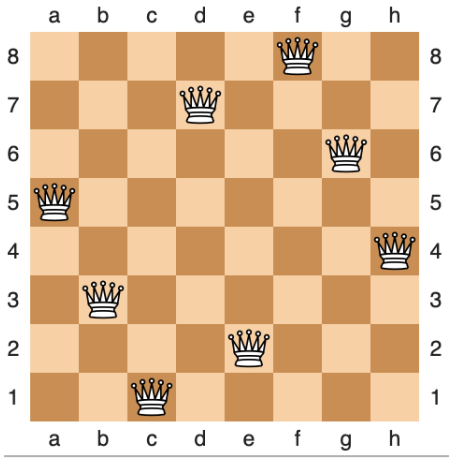
Swarat Chaudhuri

Deadline: 11:59pm, February 18, 2021

Notes

- Please submit a zip file with your code, along with a README, on Canvas. For each problem, please use a separate `.rkt` file annotated with the problem number.
- Discussions with other students is permitted, but please write your own code/text.
- You can find a Racket tutorial at `https://docs.racket-lang.org/plait/Tutorial.html`.

1. In the classic n-queens puzzle, one seeks to place n queens on an $n \times n$ chessboard such that the queens do not "attack" each other. For example, here is a solution to the problem for $n = 8$.



Here we consider a version of n-queens in which some of the queens have already been inserted. Your goal is to to use Rosette to either automatically insert the remaining queens, or to determine that this is not possible.

   (a) Let us start with 4x4 puzzles. Define a representation of 4×4 chessboards. Write a Rosette function `queens-solve-4` procedure that takes as input a partially completed 4×4 puzzle, and whose output is a solution, if one exists, and `#f` otherwise.

   (b) Generalize your puzzle data structure so that it can represent $n \times n$ puzzles. Generalize `queens-solve-4` into a routine `queens-solve` for solving $n \times n$ puzzles.

2. For this problem, I recommend starting with the code that we went through in class: `https://gist.github.com/jamesbornholt/b51339fb8b348b53bfe8a5c66af66efe`.

Consider the problem of building a simple language for testing, verifying, and synthesizing boolean circuits. We are interested in representing circuits as programs that use one of the following operations: Not, And, Or, Iff, and Xor, with their usual meaning.

Here is a Rosette encoding of this language:

```
(struct notexp (a) #:transparent)
(struct andexp (a b) #:transparent)
(struct orexp  (a b) #:transparent)
(struct iffexp (a b) #:transparent)
(struct xorexp (a b) #:transparent)
```

(a) Define an *interpreter* for the above circuit data type.

(b) A *specification* for us is a Boolean expression over symbolic inputs. Define a procedure `(ver impl spec)` that takes in a fixed circuit program and a specification over the same symbolic inputs, and checks that they are *equivalent*, i.e., produce the same output on all inputs. If they are equivalent, `ver` should return `#t`. Otherwise, the routine should a counterexample, i.e., an input on which the two functions have different outputs.

(c) Use the `choose*` construct to define a nondeterministic expression `sk` — a *sketch* — representing partially complete circuits.

(d) Define a procedure `(syn sketch spec)` that takes as input a specification `spec` and a sketch `sk` over the same symbolic inputs, and synthesizes a circuit program that is equivalent to the specification. The routine should return `#f` if no such implementation exists.

3. In this exercise, we will try to perform *exact* neural learning using symbolic tools from program synthesis. Specifically, we will model a fully connected, feedforward, ReLU neural network as a Rosette sketch and use Rosette's solver to find the right weights for this network. (If you are unfamiliar with ReLU networks, please consult one of the many expositions available online.)

(a) We start with a Rosette definition of the activation function for a single neuron: compute the dot product of inputs and weights, then apply the ReLU function. (You need to use `rosette`, rather than `rosette/safe`, as the language for this part.)

```
(define (activation inputs weights)
  (define dot (apply + (map * inputs weights)))
  (if (> dot 0) dot 0))
```

And here is a definition of function that computes the activations for an entire *layer*:

```
(define (layer inputs weight-matrix)
 (for/list ([weights (in-list weight-matrix)])
   (activation inputs weights)))
```

Using these two functions, write a Rosette implementation of a fully connected neural network with two hidden layers.

(b) Note that a neural network with unknown weights is fundamentally the same as a sketch: a procedure with symbolic parameters . Write a sketch that represents the space of fully connected ReLU networks with two inputs, k hidden layers, the i-th of which has m_i neurons (k and the m_i-s are unknown and symbolic), and one output.

(c) Now suppose your goal is to automatically discover settings for the network's weights that cause the network to discover the XOR function (i.e., on any input (x_1, x_2), the network always outputs $x_1 \oplus x_2$). You can set the hyperparameters of the network (the number of hidden layers and the number of neurons in each layer) manually or discover them automatically using Rosette. Show how to do this using Rosette's synthesis capabilities.