Homework Assignment 2
Due: October 4, 2020 at 9:00 pm

All resources for this assignment are available on `courselab.cs.princeton.edu` in the directory `/u/cos516/fall20/hw2` (information on courselab).

Total points:	100
Files provided:	`SetSolve.py`, `run.py`, `demo` directory (and other files)
Deliverables:	Please submit via TigerFile here: hw2 `hw2.pdf`: for Problems 1 – 5 `SetSolve.py`: for Problem 6

1 First-Order Logic (20 points)

1. (10 points) Consider the following functions and relations (predicates) over a domain of persons and a domain of papers:

   - isResearcher(x): is a unary relation stating that person x is a researcher
   - isReviewer(x, y): is a binary relation stating that person x is a reviewer of paper y
   - isAuthorOf(x, y): is a binary relation stating that person x is an author of paper y
   - areCoauthors(x, y): is a binary relation stating that persons x and y are coauthors
   - advisor(x): is a unary function denoting person x's PhD advisor

   Give a translation from the following English sentences to FOL formulas using the functions and relations above. (If it helps, you can also use an "equality" (=) predicate, e.g., to require that two given persons are distinct.)

   (a) Two people are coauthors if and only if they have written at least one paper together.

   (b) Every researcher is coauthors with at least one other researcher.

   (c) Some researchers write all of their papers with at least one other researcher.

   (d) If any person x coauthors a paper with another person y, then x can never review y's papers.

   (e) Every researcher has written at least one paper with their PhD advisor.

2. (4 points) Let Σ be any first-order signature including equality. Show that for any natural number n, there is a Σ-sentence G_n such that for any Σ-structure M, $M \models G_n$ if and only if U^M has *at least* n elements.

3. (6 points) Consider the following FOL formula over the signature $\Sigma = (f/1; r/2)$.

   $((\forall x.\forall y.r(x,y) \vee r(y,x)) \rightarrow (\exists x.\forall y.r(x,y)))$

   If the formula is valid, prove its validity using the semantic argument method. Otherwise, provide a falsifying interpretation and, if the formula is satisfiable, a satisfying interpretation.

2 First-Order Theories (35 points)

4. (10 points) Theory Solver

   Decide the satisfiability of the following $T_=$ formula using the congruence closure algorithm discussed in class:

   $$p(x) \wedge f(f(x)) = x \wedge f(f(f(x))) = x \wedge \neg p(f(x))$$

   Show the relevant intermediate steps in the algorithm.

   (Hint: The algorithm is described in [Bradley and Manna], Section 9.2.2, Pages 247–250.)

5. (25 points) Consider the program fragment shown below for a function `foo` with two integer inputs:

```
foo(a, b):
  x := a + b
  y := b + a
  if (x < 0)
    x := -x
  else
    y := -y
  assert(x + y = 0)
```

   We can view the operations in the program ($+$, $<$, $-$) and the constant (0) as function symbols in the signature of linear integer arithmetic Σ_{LIA}. Recall (from lecture 5) that the *standard model* of LIA interprets these functions and constants in the usual way over integers, e.g., $+$ is integer addition, etc. The theory of LIA, $\mathcal{T}_{LIA}$, is the set of Σ_{LIA}-sentences that can be proved true in the standard model.

   (a) (10 points) Write a quantifier-free formula F over Σ_{LIA} such that F is satisfiable modulo $\mathcal{T}_{LIA}$ iff there is an input to `foo` that would cause the assertion to fail.

   (b) (5 points) Exhibit a structure and interpretation that satisfies the formula F (note: a *structure*, not a $\mathcal{T}_{LIA}$-structure).

   (c) (10 points) Write a set of axioms A such that: (1) each axiom is valid modulo $\mathcal{T}_{LIA}$, and (2) F is unsatisfiable modulo the theory axiomatized by A. Justify your answer.

3 SMT Solving for Fun (45 points)

Set Game. The puzzle game Set is a card game designed by Marsha Falco in 1974. It consists of a deck of 81 unique cards that vary in four features, and for each feature there are three possible values:

- Shape: diamond, squiggle, or oval
- Number of shapes: one, two, or three
- Shading: solid, striped, or open
- Color: red, green, or purple

A `set` is a combination of three cards, such that for each of the four features, the three cards must have values that are either *all the same* or *all different.* The goal of the Set game is to find all `sets` in a given board of cards.

Figure 1 shows a sample SET board with 12 cards. The three cards in the red box form a valid `set`, because the shapes, the numbers of shapes, and colors on all three cards are the same, and the shadings are all different. Similarly, the cards in the green box also form a valid `set`. You can find more information about SET on Wikipedia (https://en.wikipedia.org/wiki/Set_(card_game)).

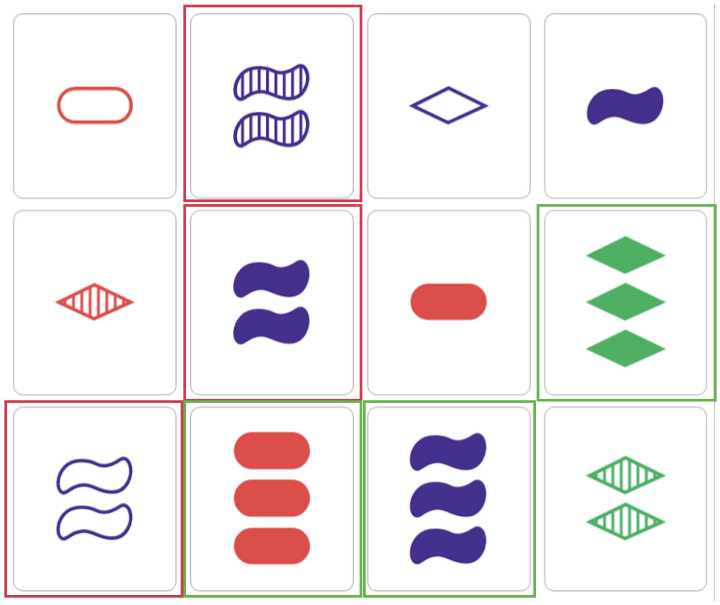

Figure 1: A sample SET board with 12 cards. The three cards in the red boxes form a `set`, and the three cards in the green boxes also form a `set`. Finding the remaining two sets on the board is left as an exercise for the reader.

6. (45 points) In this problem, we will provide you most of the code to run an interactive game of SET, where you will fill in some empty functions that implement the constraints needed to find `sets` in a given board by using the Z3 SMT-solver.

Specifically, a method `solveSet` takes a board and a list of `sets` that have already been found, instantiates the solver and adds constraints to construct a formula that is *True* if and only if a valid `set` that has not already been found[1] exists on the board. The method then runs the solver, and if the formula is satisfiable, returns a solution (a valid set), or `NONE` otherwise.

You will need to use Z3's Python API. A demonstration of how to use the API is given in `demo/php.py` and `demo/runphp.py` for the familiar Pigeon Hole Principle (PHP) problem. More information can be found in a tutorial on Z3 (http://theory.stanford.edu/~nikolaj/programmingz3.html) and other examples with Python API (http://www.cs.tau.ac.il/~msagiv/courses/asv/z3py).

The following resources are provided:

- `SetGame.py`: This file contains the python class structure for implementing the essentials for the SET game, including objects for handling a card, a deck, and a board. The `SetCard` object stores the properties for a single card and provides methods to translate to and from the format used by the Z3 solver. It also overrides comparison operators. You do not need to modify this file.
- `run.py`: This script lets users play a game of SET. Running the script with no arguments will create a new deck, then run the interactive game player. Players can guess candidate sets, and when they

[1] For our game, we will allow overlapping SETS, i.e., different sets can share cards, but not all three cards.

have found all of the sets on the board (or they give up), the next board will be dealt. Alternatively, running the script with an argument containing a path to a board file will load that single board and allow it to be played. Again, you do not need to modify this script file.

- `testSolver.py`: This file provides a small test case for your code.
- `sample_board`: A sample SET board is shown in the format expected by the code.
- `SolveSet.py`: This file contains the method `solveSet` for finding the next `set` on the board.

  The following parts in this file are provided:

  - getSetMatrix: this method creates a matrix to represent the cards and their properties.
  - matrixVar: the elements of the matrix are variables, each of which represents one feature of a particular card.
  - solveSet: this method does the boilerplate work of creating a solver, calling the constraint functions, and returning the result of running the solver.
  - addValueConstraints: this method is included as an example of how to write constraints for SET. This constraint encodes the rule that the returned set contains cards with valid values as their features.

  The SolveSet.py has the following empty functions that you will need to implement using Z3's Python API:

  1. `def addDuplicateConstraints(X)`: add constraints to ensure that there are no duplicate cards in the set.
  2. `def addBoardConstraints(X, board)`: add constraints to ensure that all cards in the set are on the board.
  3. `def addPrevFoundSetsConstraints(X, found_sets)`: add constraints to ensure the set hasn't been found already.
  4. `def addValidSetConstraints(X)`: add constraints to ensure the set is valid, i.e., for each feature, all three cards are the same or all three are different.

  In the above rules, `X` is a matrix that represents a candidate `set` with three cards, `board` represents the board of cards, and `found_sets` represents the `sets` that have already been found.

To test your code, do the following:

- Set a global shell variable `PYTHONPATH` to `/u/cos516/z3/build`.
  - bash: `export PYTHONPATH=/u/cos516/z3/build`
  - cshell: `setenv PYTHONPATH /n/cos516/z3/build`
- Now, you can run the following two commands:
  - `python3 testSolver.py`: This is a small test on the sample board of cards.
  - `python3 run.py`: This is the main file to play an interactive game of SET after implementing the above methods. Give it a try!

The interactive game allows a *hint* command, which uses the Z3 solver to suggest a card that is part of a valid undiscovered `set`. If the user asks for multiple hints in a row, additional cards in that `set` will be revealed. At the end of the game, the user's score is displayed, consisting of the number of `sets` found, the total number of `sets` that existed in the board, and the number of hints used.

Acknowledgements. Thanks to Anne Kohlbrenner and Ben Kaiser for designing and implementing the game of SET using Z3 for this course homework.