CSC A48 Assignment 2

Formula Game

Due: Saturday, March 10, 2018 at 5pm.

⋆ introduction

◯ boolean variable (or just variable)

A boolean variable is a variable that takes on one of two values, True and False. For convenience, we will use 1 and 0 to denote True and False respectively. Also, all our variable names will consist of a single lower case letter. So we are limited to at most 26 variables.

◯ boolean formula (or just formula)

A boolean formula is a string made up of the following.

- Boolean variables. I.e., lower case letters.
- The connectives $\wedge$, $\vee$, $\neg$, called AND, OR, NOT respectively. For ease of typing a formula on a keyboard, we will use the characters `*`, `+`, `-` to denote the symbols $\wedge$, $\vee$, $\neg$ respectively.
- Left and right parentheses. I.e., the symbols `(` and `)`.

⊙ rules for formulas

Here are the rules governing what constitutes a valid boolean formula.
Note the recursive nature of these rules.

1. The simplest formula is a string consisting of just one variable.
2. If F_1 and F_2 are formulas, then so are $(F_1 * F_2)$, $(F_1 + F_2)$, and $-F_1$.
   Using Python syntax, we mean the following strings are formulas.
   - `'('` + F_1 + `'*'` + F_2 + `')'`
   - `'('` + F_1 + `'+'` + F_2 + `')'`
   - `'-'` + F_1

   Use of parentheses is absolutely required for the first two cases, and not permitted for the last case. These restrictions may seem overly cumbersome, but you'll thank us when we ask you to write code to work with these formulas! ⌣̈
3. Any string that cannot be constructed using the above rules is not a valid formula.

⊙ examples

Here are some valid formulas (shown as Python strings).

- `"x"`
- `"-y"`
- `"(x*y)"`
- `"((-x+y)*-(-y+x))"`

Here are some strings that are not formulas.

- `"X"` *variable not lower case letter*
- `"x*y"` *missing parentheses*
- `"-(x)"` *extraneous parentheses*
- `"(x+(y)*z)"` *mismatched parentheses*

◯ tree representation of a formula

Boolean formulas can be represented by a binary tree in a very natural way. Each leaf node represents a variable. Each internal node represents a connective, called an AND node, an OR node, or a NOT node, depending on the connective represented. A node that represents `*` or `+` has 2 child nodes, which in turn represent the two subformulas that are ANDed or ORed to form the larger formula. A node that represents `-` has one child node, which in turn represents the formula that is NOTed. We draw these trees in a way that requires us to rotate it 90 degrees clockwise to see it with its root at the top. Alternatively we could also turn our head toward our left shoulder. We also draw them without lines (or arrows) betweeen parent and child nodes. As with Python code, parent child relationships can be inferred by the level of indentation. Children of the root are indented 2 spaces. Children of a child of the root are indented 4 spaces, and so on. For example, here is the tree for the formula `((-x+y)*-(-y+x))`.

```
* - + x
      - y
  + y
    - x
```

This may seem an awkward way to draw a tree, but you'll thank us when we ask you to write code to draw such trees! ☺

Important: Make sure you understand how formulas are represented by trees. Try writing down a formula, then drawing its corresponding tree. Repeat until it becomes easy for you.

◯ truth value of a boolean formula

If we assign a truth value (1 or 0) to each variable in a formula, then we can determine the truth value of that formula in a natural way. Formally, here the rules governing the truth value of a formula.

1. For a formula consisting of just one variable, the truth value of the formula is the truth value of the variable.

2. For a formula constructed from smaller formulas F_1 and F_2, its truth value is as follows.

   - The truth value of a formula $(F_1 * F_2)$ is 1 if both F_1 and F_2 have truth value 1, and 0 if at least one of F_1 and F_2 has truth value 0. Thinking of 1 and 0 as integers, we could say that the truth value of $(F_1 * F_2)$ is the minimum of the truth values of F_1 and F_2.
   - The truth value of a formula $(F_1 + F_2)$ is 1 if at least one of F_1 and F_2 has truth value 1, and 0 if both F_1 and F_2 have truth value 0. Thinking of 1 and 0 as integers, we could say that the truth value of $(F_1 + F_2)$ is the maximum of the truth values of F_1 and F_2.
   - The truth value of a formula $-F_1$ is 1 if F_1 has truth value 0, and 0 if F_1 has truth value 1. Thinking of 1 and 0 as integers, we could say that the truth value of $-F_1$ is one minus the truth value of F_1.

◯ **formula game**

The formula game is a game for two players, called A and E. The game starts with a boolean formula F, along with an ordering of the variables that appear in F, and for each variable, which player gets to choose its truth value. The players take turns choosing the values of the variables one at a time in the specified order. After all the variables in F have been assigned truth values, player A wins if the truth value of F is 0, and player E wins if the truth value of F is 1. I.e., player A is trying to make F false, and player E is trying to make F true.

⊙ **example**

Consider the following formula, along with information regarding which player gets to choose which variable's truth value in which order.

formula:	`((x+y)*((y+z)*(-y+-z)))`
turns:	`EEA`
variables:	`xyz`

For this game, player E first chooses the value of `x`, then the value of `y`. Finally player A chooses the value of `z`. If player E chooses 0 for `x` and 1 for `y`, and player A chooses 1 for `z`, then player A wins. On the other hand, if player E chooses 1 for `x` and 0 for `y`, and player A chooses 1 for `z`, then player E wins.

Please work through the above scenarios and be certain that you understand them.

◯ **formula game configuration**

We represent the state of a formula game as it is being played by 4 pieces of information.

formula:	the formula, possibly represented by a formula tree
turns:	a string of A's and E's, indicating who takes which turn
variables:	a string of the variables in the formula, in order as they are assigned values
values:	the values of the variables chosen so far

⊙ **example**

From the previous example, here is the initial configuration.

formula:	`((x+y)*((y+z)*(-y+-z)))`
turns:	`EEA`
variables:	`xyz`
values:	(empty string)

Here is the configuration after the first turn, supposing player E chooses 0 for `x`.

formula:	`((x+y)*((y+z)*(-y+-z)))`
turns:	`EEA`
variables:	`xyz`
values:	`0`

Here is the configuration after the second turn, supposing player E chooses 1 for `y`.

formula:	`((x+y)*((y+z)*(-y+-z)))`
turns:	`EEA`
variables:	`xyz`
values:	`01`

Here is the final configuration after the third and last turn, supposing player A chooses 1 for `z`.

formula:	`((x+y)*((y+z)*(-y+-z)))`
turns:	`EEA`
variables:	`xyz`
values:	`011`

⋆ what to do

Your task consists of multiple parts.

◯ design part

Create a UML diagram for the design of a collection of classes to represent the various sorts of formula trees. Each class should implement or inherit an `__eq__` method so that it can be compared to other objects; and a `__repr__` method so that it can be represented in a meaningful way as a string, and so that an equivalent tree can produced if you cut and paste the representation into a Python shell. Be sure to include these methods in your diagram. You should carefully consider how to use inheritance to reduce the amount of duplicated code.

◯ programming part

Here are the steps.

1. Download the files `formula_tree.py`, `formula_game_functions.py`, and `play_formula_game.py`.
   For the file `formula_game_functions.py`, add your name to the license at the top to indicate that you have added intellectual value to it. However, do not add any `import` statements, or change the `import` statement.
   You may only distribute these files, or modified versions of them, with the same license, and along with the file `GNUlicense`.

2. In `formula_game_functions.py`, implement the following module-level functions, as usual with good documentation.

   `build_tree(formula)`: which takes a string `formula` and returns the `FormulaTree` that represents `formula` if it is a valid formula. Otherwise `None` is returned.

   `draw_formula_tree(root)`: which takes the `FormulaTree` rooted at `root` and returns the string that draws that tree (as described on page 2).

   `evaluate(root, variables, values)`: which takes a formula as represented by the `FormulaTree` rooted at `root`, along with a string `variables` containing the variables in the formula and a string `values` (of 1's and 0's) containing the corresponding truth values for the variables, and returns the truth value (1 or 0) of the formula.

   `play2win(root, turns, variables, values)`: which takes the formula game configuration given by `(root, turns, variables, values)`, and returns the best next move for the player whose turn is next. The formula part of the configuration is represented by the `FormulaTree` rooted at `root`.
   Implement this so that if there is a winning strategy for that player, then the corresponding "winning" move is returned. If there is no winning strategy, or if choosing either 1 or 0 would lead to winning, then 1 is returned if it is player E's turn, and 0 is returned if it is player A's turn. `(root, turns, variables, values)` must form a valid formula game configuration, and length of `turns` must be greater than length of `values` — i.e., there must be a next move.

3. Run the program `play_formula_game.py` and enjoy playing the game!

◯ things to think about part

- If we were to allow formulas of the form $(F_1 \odot F_2 \odot \cdots \odot F_k)$, where $\odot$ is either `*` or `+`, $k > 1$, and each F_i is a formula, then the corresponding AND or OR node would have k child nodes. For example, for the formula `(x+(u*v*w*z)+y)`, its tree would be drawn as follows.

```
+ y
  * z
    w
    v
    u
  x
```

  Your functions that take `root` as a parameter, would they also work with trees having such nodes? If not, then try to make them do so.

- Further to the above, try writing a function `simplify_tree(root)`, which takes the formula tree rooted at `root` and modifies it so that no AND node is the child of an AND node, and no OR node is the child of an OR node. For example, "simplifying" the tree that represents the formula `(x*(y*z))` should result in the tree that represents the formula `(x*y*z)`.

- Try writing a function `tree2formula(root)`, which takes the formula tree rooted at `root` and returns the formula represented by the tree. Would your `tree2formula` work with a "simplified" tree as described above?

You may submit these functions as part of your `formula_game_functions.py`, but they will not be marked. However, take special care to ensure that these extra functions do not *break* (affect adversely) any function that will be marked.

◯ what to submit

Submit your `formula_game_functions.py` file to Markus.