Logic and Computability SS23, Assignment 2

Due: 23. 03. 2023, 23:59

Please pull (or download) the skeleton files from the upstream repository in order to get started:

```
git pull upstream main
```

If you do not want to use git, you can download the two skeleton files directly from the server:

https://git.pranger.xyz/sp/LAC-Practical-Assignments-2023/src/branch/main/Assignment2

If you download the files manually, please create a folder `Assignment2` in your local repository.

Please also install colorama:

```
pip install colorama
```

The assignment sheet for this week is split into two tasks. For the first task, you will encode the map colouring problem via z3. For the second task, we will make use of z3's uninterpreted functions to compute a seating arrangement for a wedding.

1. [7 Points] **Map Colouring.** For the first task you need to compute solutions to the map colouring problem. You are given an $m \times n$ grid consisting of multiple regions, each represented by a unique character.

   The task is to colour all cells of a region with the same colour, while adjacent regions need to be coloured with different colours. You are allowed to use four colours to solve this task.

```
ABBBBBBBB
ACCCCCCCB
ACDEEEECF
ACDGHHECF
ADDGGIECF
ADIIIIFCF
ADJKKLFFF
AJJJKLLLL
AAMMMMMMM
```

Figure 1: An example map with 13 regions.

   The parsing of the input map and splitting the map into regions is handled for you in the skeleton file `colours.py`. Continue with implementing the following task to solve the problem:

   `Subtask` 1. [2 Points]

   (a) Create a Z3 `Datatype` that represents colours. Populate it with four colours of your choice.

   (b) Create a Z3 sort from your datatype.

   (c) Create a Z3 variable of your sort for each cell in the playground. You should give them useful names, like `C_0_0`, `C_2_3` to make debugging easier.

   `Subtask` 2. [3 Points] Enforce that cells of the same region have the same colour. To to so, use the transitivity property of equality, i.e. if $c_1 = c_2 \wedge c_2 = c_3 \rightarrow c_1 = c_3$.

   `Subtask` 3. [2 Points] Enforce that neighbouring cells from different regions need to have different colours.

2. [8 Points] **Seating Arrangement Problem.** For the second task of this week, we take a look at the seating arrangement problem. The problem that you are facing is the following: There are only a few hours left before your weeding and you have lost the seating plan for the big table. You and your friends have gathered a list of all to be seated at the big table, but you cannot simply place them in any order since this could lead to some unpleasant situations. Alongside the list of members at the table you have also come up with a pairs of people which need to be seated next to each other and some which need to have some other guests in between.

   The input that you have come up with is a list of:

   - A pair of friends: `Bob likes Alice`,
   - a pair of foes: `Ada dislikes Bob`,
   - a guest without preferences: `John` or
   - a comment: `#John likes Ada`.

   For this exercise, we will make use of Z3's `uninterpreted functions` and `uninterpreted sorts`.

   The parsing of the list is handled for you in the skeleton file `seating-arrangement.py`. Continue with implementing the following task to solve the problem:

   `Subtask` 1. [1 Points] Extend the parsing by adding pairs of friends and foes to their respective lists.

   `Subtask` 2. [1 Points] Define a Z3 uninterpreted function that maps `Guests` to positions at the table.

   `Subtask` 3. [2 Points] Define a function `neighbours` which returns whether two guests are neighbours.

   `Subtask` 4. [2 Points] Enforce that all guests are seated at the table, and that no two guests will be seated at the same place.

   `Subtask` 5. [2 Points] Enforce that friends need to be seated next to each other and foes must not be seated next to each other.

   We represent the positions at our table with an uninterpreted function which maps to the integer positions at the table. Note that the table wraps around on both ends of this integer list that we are mapping to, i.e. position `0` is right next to position `len(guests)`.

   If we can find a proper seating plan which adheres to our given constraints we will visualize it, otherwise Z3 will tell us that the constraints are not satisfiable. Note that Z3 will not fully define the uninterpreted function, which means that we need to extend the call to `model.evaluate(...)` to `model.evaluate(...,model_completion=True)`. The seating plan will be visualized with a table in your terminal and as a list of the mapping in the very end of our program call.

```
Patric dislikes Ada
Patric dislikes Katie
Patric dislikes Bob
Ada
John likes Alice
Bob likes Andrea
Andrea
Alice
Ada likes Julia
Ada likes Katie
Robert
```

Figure 2: The constraints and member list that you have come up with.

```
       Patric Robert
       ---------------------
Andrea|<<<<<<<<<<<<<<<<<<<<<|Katie
      |>>>>>>>>>>>>>>>>>>>>>|
   Bob|<<<<<<<<<<<<<<<<<<<<<|Ada
      |>>>>>>>>>>>>>>>>>>>>>|
  John|<<<<<<<<<<<<<<<<<<<<<|
       ---------------------
       Alice Julia

Seating plan:['Patric', 'Robert', 'Katie', 'Ada', 'Julia',
'Alice', 'John', 'Bob', 'Andrea']
```

Figure 3: A possible seating plan for the given input file from Figure 1.