Logic and Computability SS23, Assignment 3

Due: 30. 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 skeleton file directly from the server:

https://git.pranger.xyz/sp/LAC-Practical-Assignments-2023/src/branch/main/Assignment3

If you download the file manually, please create a folder `Assignment3` in your local repository.

The last programming assignment sheet consists of a single exercise. This week we are going to play **Rock-Paper-Scissors-Spock-Lizard** against the computer. Your task is to collect knowledge about the *pseudo-random number generator* that decides the choice of the computer to predict its next choice.

1. [15 Points] **Rock-Paper-Scissors-Spock-Lizard**

   For this task you need to model the computation of a *linear-congruential generator*. If you want to read more about linear-congruential generators, please consult the wikipedia.

   Our opponent, the computer, computes its choice by first computing a pseudo-random number s_i that is based on the previous result s_{i-1}:

$$s_i = 11 \cdot s_{i-1} + 12345 \;\&\; \text{0x7fff}. \tag{1}$$

   Its choice c_i is then computed with the remainder modulo 5:

$$c_i = s_i \mod 5, \tag{2}$$

   where

   - $c_i = 0$ means Rock,
   - $c_i = 1$ means Paper,
   - $c_i = 2$ means Scissors,
   - $c_i = 3$ means Spock, and
   - $c_i = 4$ means Lizard.

   Your task is to model the computation of each s_i, starting from s_1, and use your knowledge about the computer's choice to tell Z3 that $c_i == s_i \mod 5$. Note that you do not know the value of s_0 in this scenario. (The problem would become trivial if you would have that information!)

   The skeleton code can be found in `rpssl.py`. The snippet contains a class `RPSSLComputer` that reads a seed value (`s_0`) and continuously returns the next computer's choice by calling `compute_choice()`.

   You execute the script by passing it a value for the seed:

```
python3 rpssl.py <seed>.
```

   After you have implemented the correct constraints, your script will win continuously after a few rounds, independently of the seed value.

Implementation

The snippet simulates the game against the computer. For all rounds $j \in \{1, \dots, i\}$ that you have already played you can give Z3 the information that s_j has been computed via Eq. 1 and the computer's choice c_j via Eq. 2. Adding a constraint for Eq. 1 for the current round $i + 1$ will make Z3 compute a good guess for s_{i+1}. You can use this result to come up with a winning move against the computer.

The script starts with querying the computer for `preprocess_count` many choices and calls a function (`add_constraint(…)`) where you add the appropriate constraints to the solver. These constraints will describe the relationship between s_i and s_{i-1} and the relationship between s_i, c_i and the computer's choice.

After this preprocessing is done, you are going to use Z3 to compute potential good choices for the current round $i + 1$ in the game. In order to ask Z3 for a good choice you will model the computation that the computer will make for s_{i+1} and add this to the solver in the function `add_next_state_constraint(…)`.

You can then use the resulting value of s_{i+1} from the model and compute the remainder modulo 5 to beat the computer. Implement this as the return value of `get_players_choice(…)` using the `winning_mapping`.

The special variable `s_i_plus_1` will always hold the value for the current round. After you have extracted the value for s_{i+1} for the current round, you need to remove the constraint for this variable and add the constraints about the round that you have already played, using `add_constraint(…)`. Note that *this loop is already implemented for you* and uses `store_backtracking_point(…)` and `restore_backtracking_point(…)` to correctly add and remove the constraints for `s_i_plus_1`.

You are going to solve the task by filling in the function bodies of three different functions:

Subtask 1. [7 Points] `add_constraint(…)`
In this function we give Z3 our knowledge about the rounds we have played so far.

(a) Create a Z3 BitVector that stores 16 bits and append this to the `states` list.
(b) Enforce that the newly added state evaluates to the LCG computation (1) of the previous state.
(c) Enforce that the unsigned remainder modulo 5 of the newly added state is equal to the computer's choice.
Hint: Use the built-in function URem(s_i, 5) to model the remainder.

Subtask 2. [5 Points] `add_next_state_constraint(…)`
In this function we tell Z3 that the computer will compute s_{i+1} via 1:

(a) Enforce that `s_i_plus_1` evaluates to the LCG computation (1) of the previous state.

`Subtask` 3. [3 Points] `get_players_choice(…)`

(a) Use the model that the solver came up with and `winning_mapping` to decide the your choice for the next round of the game.