Spectra Example: Cinderella-Stepmother Problem

Ilia Shevrin and Matan Yossef

Tel Aviv University

Abstract. The Cinderella-Stepmother problem is a two player game in which Cinderella fights her stepmother for keeping the safety of a water buckets system. This report presents a formal specification of the problem in the Spectra language. We model it as a safety game between Cinderella as the system and the stepmother as the environment.

1 Introduction

The Cinderella-Stepmother problem is a two player game in which Cinderella fights her stepmother for keeping the safety of a water buckets system [1]. The starting position of the game consists of `N` empty water buckets that are arranged in a circle. Each bucket has a capacity of `B` water units. In each turn, Cinderella's stepmother pours `A` water units into the buckets in a distribution of her choice. Followed by this, Cinderella empties `C` adjacent buckets. Cinderella's stepmother again pours `A` water units into the buckets and so on. As Cinderella's stepmother tries to entirely fill one of the buckets, i.e., fill it with `B` water units, Cinderella plays a safety game of avoiding it.

The goal is to describe a behavior for Cinderella such that her stepmother will not be able to fill any of the buckets entirely, i.e., each will never contain `B` water units, assuming an unlimited supply of water for Cinderella's stepmother.

We present a Spectra [4] specification for the problem and a simulation of its solution. All related files are available from `http://smlab.cs.tau.ac.il/syntech/`.

2 The Specification

We model the problem as a parametric specification that depends on the following parameters:

- `N` represents the number of water buckets in the system.
- `B` represents the maximum number of water units a bucket can contain.
- `A` represents the number of water units the stepmother pours into the buckets in each turn.

[1] Variants of the problem have appeared in the literature, e.g., [1,2,3]. One previous formulation of the problem appears in the git repository of the Slugs synthesizer `https://github.com/VerifiableRobotics/slugs`

- `C` represents the number of adjacent buckets Cinderella empties in each turn.

```
define N := 5;
define C := 2;
define B := 9;
define A := 5;
```

Listing 1.1. Spectra specification of the Cinderella-stepmother problem: defines of problem parameters

As we will see later, the realizability of the problem depends on the configuration of the parameters. For now, we will use the parameters values that are defined in List. 1.1.

The model refers to the wicked stepmother's water supply as the environment. Meanwhile, Cinderella acts as the system and tries to keep the buckets' state safe. The variable declarations are shown below in List. 1.2.

```
env Int(0..A)[N] unitsToFill;

sys Int(0..B)[N] buckets;
sys Int(0..(N - 1)) firstToEmpty;

predicate inAdjacentBucketsToEmpty(Int(0..(N-1)) i):
  (i >= firstToEmpty & i < firstToEmpty + C) |
  i < firstToEmpty + C - N;
```

Listing 1.2. Spectra specification of the Cinderella-stepmother problem: declaration of system and environment variables and a useful predicate

Each step in the game consists of stepmother's choice of dividing `A` units of water between `N` buckets, represented by the environment array `unitsToFill`, followed by Cinderella's choice of emptying `C` adjacent buckets, represented by the system integer `firstToEmpty`. The buckets array is a system array represented by `buckets`.

We define a useful predicate, `inAdjacentBucketsToEmpty(i)`, which represents a bucket that is going to be emptied, i.e., it belongs to the adjacent sub array that Cinderella chose. More formally, bucket with index i is emptied iff it holds that $\texttt{firstToEmpty} \leq i < \texttt{firstToEmpty} + \texttt{C}$ or $i < \texttt{firstToEmpty} + \texttt{C} - \texttt{N}$ to account for the cyclic nature of the buckets array. See List. 1.2.

```
gar forall i in Int(0..(N - 1)). buckets[i] = 0;
```

Listing 1.3. Spectra specification of the Cinderella-stepmother problem: initial state guarantees

Initially, the buckets are empty (see List. 1.3). We assume that the stepmother fills exactly `A` water units on her turn (see List. 1.4). We use a Spectra construct that allows us to conveniently define numeric operations on arrays. In this case, `sum` syntactically translates to the sum of all the elements in the array.

```
asm alw unitsToFill.sum = A;
// Translates to : asm alw unitsToFill[0] + unitsToFill[1] +
    unitsToFill[2] + unitsToFill[3] + unitsToFill[4] = A;
```

Listing 1.4. Spectra specification of the Cinderella-stepmother problem: stepmother assumption

The next two guarantees, shown in List. 1.5, specify the next state of the buckets array following the stepmother's decision of which buckets to fill combined with Cinderella's decision of which buckets to empty. Specifically, the next state value of an emptied bucket *i* is the quantity that the stepmother chooses to fill into this bucket on the next step (`next(buckets[i] = next(unitsToFill[i]))`). The next state value of a non-emptied bucket is its current value plus the quantity that the stepmother chooses to fill into this bucket on the next step (`next(buckets[i] = buckets[i] + next(unitsToFill[i]))`).

```
gar G forall i in Int(0..(N - 1)).
 inAdjacentBucketsToEmpty(i)
     -> (next(buckets[i]) = next(unitsToFill[i]));

gar G forall i in Int(0..(N - 1)).
 !inAdjacentBucketsToEmpty(i)
   -> (next(buckets[i]) = buckets[i] + next(unitsToFill[i]));
```

Listing 1.5. Spectra specification of the Cinderella-stepmother problem: Cinderella's guarantees

3 Using the Specification

3.1 Realizability

We used Spectra to check the realizability of the problem in different configurations. We see some interesting results, where some are perhaps less intuitive than others.

Table 1. Realizabily Results for `N`=5 Buckets

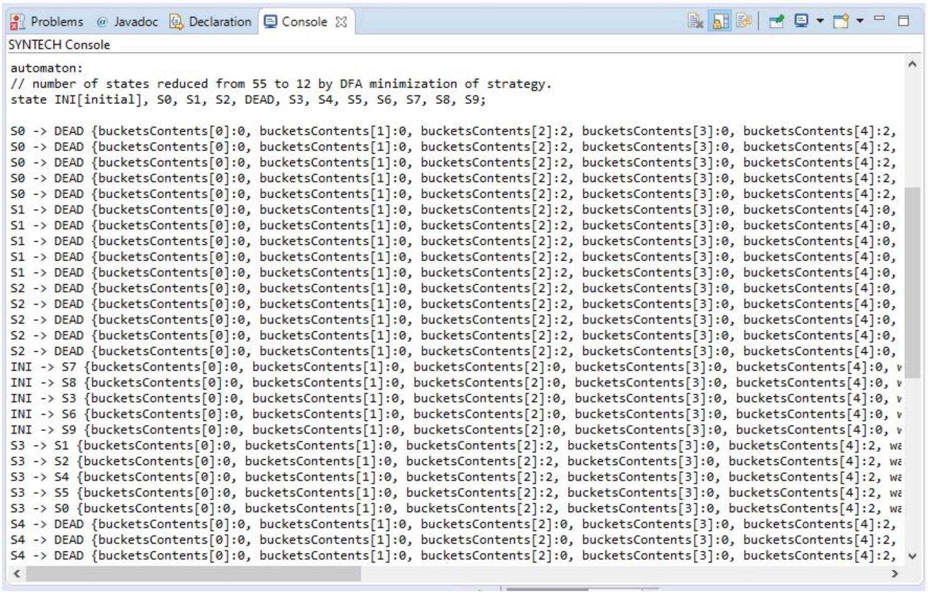
`B` (bucket capacity)	`A` (added units)	`C` (adjacent buckets to empty)	Realizable?
6	4	2	N
7	4	2	N
8	4	2	Y
6	4	3	N
7	4	3	Y
9	5	2	N
10	5	2	Y

Realizability checking does not provide any information about how can Cinderlla keep the system safe (when the specification is realizable) or how can the wicked stepmother fill one of the buckets entirely (when the specification is unrealizable). For these, we use two other useful Spectra tools as follows.

3.2 Counter-strategies

Spectra allows the engineer to synthesize a concrete counter-strategy. For the unrealizable configurations, we can create a controller that shows how the stepmother can fill one of the buckets entirely in response to any sequence of Cinderella's moves.

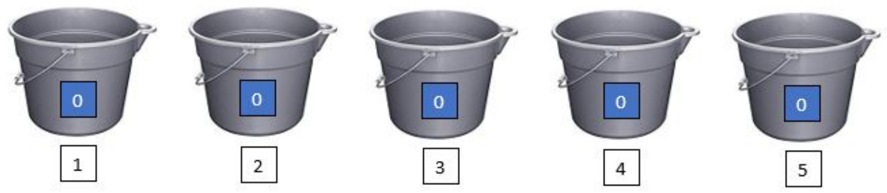
For example, consider the configuration of 5 buckets with capacity of 6 water units, where the stepmother can add 4 water units at each step and Cinderella can empty 2 adjacent buckets as a response. This configuration of the specification is unrealizable. After synthesizing a counter-strategy, Spectra outputs a textual representation of the controller to the console, similar to the one below.



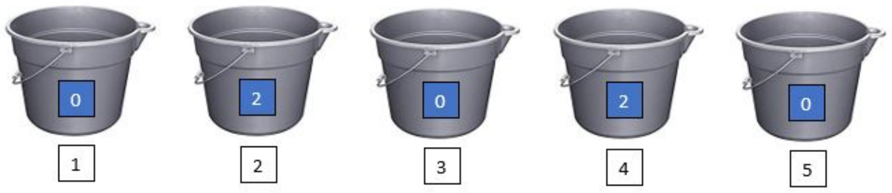
```
automaton:
// number of states reduced from 55 to 12 by DFA minimization of strategy.
state INI[initial], S0, S1, S2, DEAD, S3, S4, S5, S6, S7, S8, S9;

S0 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2,
S0 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2,
S0 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2,
S0 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2,
S0 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2,
S1 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
S1 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
S1 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
S1 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
S1 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
S2 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
S2 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
S2 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
S2 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
S2 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:0,
INI -> S7 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:0, bucketsContents[3]:0, bucketsContents[4]:0, v
INI -> S8 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:0, bucketsContents[3]:0, bucketsContents[4]:0, v
INI -> S3 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:0, bucketsContents[3]:0, bucketsContents[4]:0, v
INI -> S6 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:0, bucketsContents[3]:0, bucketsContents[4]:0, v
INI -> S9 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:0, bucketsContents[3]:0, bucketsContents[4]:0, v
S3 -> S1 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2, wa
S3 -> S2 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2, wa
S3 -> S4 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2, wa
S3 -> S5 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2, wa
S3 -> S0 {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:2, bucketsContents[3]:0, bucketsContents[4]:2, wa
S4 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:0, bucketsContents[3]:0, bucketsContents[4]:2,
S4 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:0, bucketsContents[3]:0, bucketsContents[4]:2,
S4 -> DEAD {bucketsContents[0]:0, bucketsContents[1]:0, bucketsContents[2]:0, bucketsContents[3]:0, bucketsContents[4]:2,
```

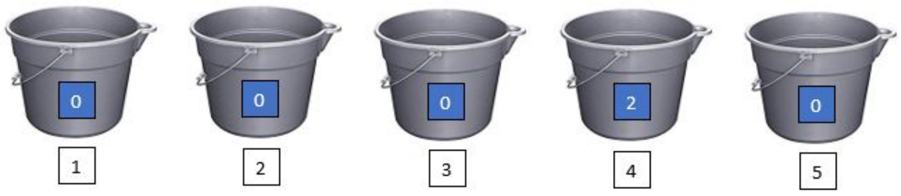
If we follow the controller's states, starting from the INI state to all the DEAD states, we can get a grab at a counter-strategy that the stepmother can use to fill one of the buckets entirely. We visualize one such counter-strategy below:



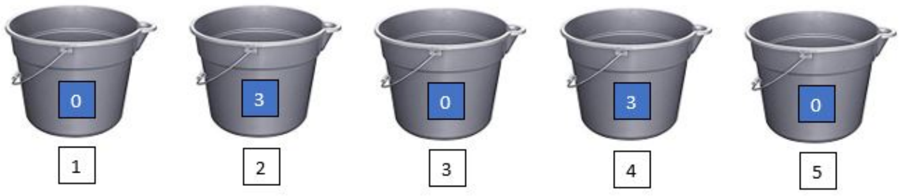
The **stepmother** first pours 2 water units into two buckets that are separated by at least one bucket, e.g., buckets 2 and 4:



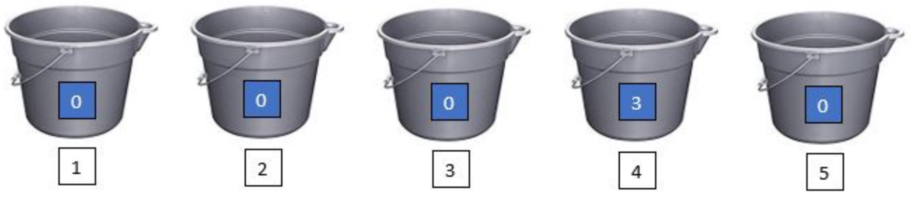
Cinderella can empty only one of the buckets (since it can empty only two adjacent buckets). W.L.O.G. it empties bucket number 2:



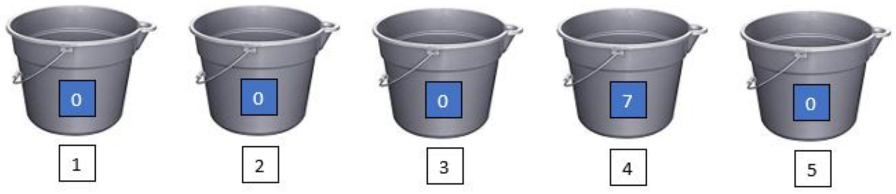
Then, the **stepmother** pours 1 water unit to bucket 4 and 3 water units to bucket 2:



Again, **Cinderella** can empty only one of the buckets (since it can empty only two adjacent buckets). W.L.O.G. it empties bucket number 2:



The **stepmother** pours 4 water units to bucket 4 to fill it entirely and win:



3.3 Cinderlla-Stepmother Simulation

To simulate the Cinderella-Stepmother problem, download the zipped simulation project and load it to an eclipse workspace. This simulation simulates the Cinderella-Stepmother game with 5 buckets.

To use the simulation, take the following steps:

1. Choose parameter values for the problem - the number of buckets Cinderella can empty in one turn, the number of water units the stepmother can add in one turn and the capacity of the buckets. Edit the `bucketsToEmpty`, `addedWaterUnits`, and `capacity` parameters of `Board.java` in the `src` folder accordingly to match the values you chose.
2. In `CinderellaStepmotherN5.spectra` edit the defines `C` (number of buckets Cinderella can empty), `B` (bucket capacity) and `A` (number of water units the stepmother adds) with the parameter values you chose and synthesize a symbolic controller for the specification.
3. Run the java simulation.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 638049, SYNTECH).

References

1. T. A. Beyene, S. Chaudhuri, C. Popeea, and A. Rybalchenko. A constraint-based approach to solving games on infinite graphs. In *POPL'14*, pages 221–234. ACM, 2014.
2. M. H. L. Bodlaender, C. A. J. Hurkens, V. J. J. Kusters, F. Staals, G. J. Woeginger, and H. Zantema. Cinderella versus the wicked stepmother. In *TCS'12*, volume 7604 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2012.
3. A. Katis, G. Fedyukovich, H. Guo, A. Gacek, J. Backes, A. Gurfinkel, and M. W. Whalen. Validity-guided synthesis of reactive systems from assume-guarantee contracts. In *TACAS'18*, volume 10806 of *Lecture Notes in Computer Science*, pages 176–193. Springer, 2018.
4. S. Maoz and J. O. Ringert. Spectra: A specification language for reactive systems. *CoRR*, abs/1904.06668, 2019.

A Complete Specification

```
spec Cinderella

define N := 5;
define C := 2;
define B := 9;
define A := 5;

env Int(0..A)[N] unitsToFill;

sys Int(0..B)[N] buckets;
sys Int(0..(N - 1)) firstToEmpty;

predicate inAdjacentBucketsToEmpty(Int(0..(N-1)) i):
  (i >= firstToEmpty & i < firstToEmpty + C) |
  i < firstToEmpty + C - N;

asm alw unitsToFill.sum = A;
// Translates to : asm alw unitsToFill[0] + unitsToFill[1] +
    unitsToFill[2] + unitsToFill[3] + unitsToFill[4] = A;

gar forall i in Int(0..(N - 1)). buckets[i] = 0;

gar G forall i in Int(0..(N - 1)).
  inAdjacentBucketsToEmpty(i)
     -> (next(buckets[i]) = next(unitsToFill[i]));

gar G forall i in Int(0..(N - 1)).
  !inAdjacentBucketsToEmpty(i)
   -> (next(buckets[i]) = buckets[i] + next(unitsToFill[i]));
```