Safe Population Growth with Rule 30

Md Solimul Chowdhury, Martin Müller, Jia-Huai You
Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada.
{mdsolimu, mmueller, jyou}@ualberta.ca

***Abstract*—A *population* is an one-dimensional grid of $n \geq 1$ organisms, where each organism evolves between being alive (1) and dead (0) across chronological time steps by following a fixed rule of evolution. At any time step $t \geq 1$, the combined states of n organisms represent the state of the population at t. At t, a population is under the *threat of extinction*, if the number of alive organisms falls below $n * (P/100)$, where $0 < P \leq 100$ and *safe*, otherwise. We say that a population *grows* over T time steps, if for any time step $t < T - 1$, population at t' has more alive organisms than population at t, with $t' = t + 1$.**

In our proposed SAT benchmark *Safe Population Growth (SPG)*, given a population of n organisms, a maximum time step T, we verify if a population could *safely grow* upto time step T, while following a fixed rule of evolution. For the SAT competition 2021, we have submitted 20 instances of the SPG benchmark.

I. SPG as a Cellular Automaton

State evolution in the Safe Population Growth (SPG) problem represents the state evolution of cells in finite elementary cellular automaton (CA) [2], with respect to (i) the *safety* constraint at any given time step and (ii) the *growth* constraint between any two consecutive time steps.

In an elementary CA, at time step $t + 1$, the state of a cell c, which has cell l (resp. r) as its left (resp. right) neighbour, is computed based on a boolean combination the states of c, l,and r at time t. There are $2^3 = 8$ combinations of boolean values for l, c, and r at t, for each of which, there are 2 ways to set the value of the state of c at $t + 1$. Hence, there are $2^8 = 256$ ways to set the new state of the c at $t + 1$. Each of these 256 ways are called *rules* [2] for a given elementary CA.

We consider *Rule 30* [3] for the SPG problem, which is known to produce chaotic patterns over time. At time $t + 1$, for a given center cell (*center*), its left (*left*) and right (*right*) neighbours, Rule 30 computes the state $center^{t+1}$ of the *center* cell as follows:

$$center^{t+1} \leftarrow left^t \ \mathbf{XOR} \ (center^t \ \mathbf{OR} \ right^t)$$

.

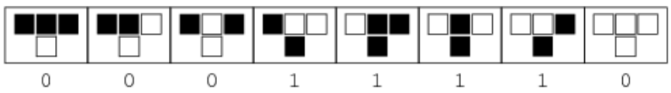

Fig. 1: State evolution for the center cell for Rule 30; black cells represents alive (1) cells, white cells represents dead (0) state.

Figure 1 (taken from [3]) shows the evolution scheme for Rule 30, which is known to exhibit chaotic behavior for some initial states. Figure 2 shows such a chaotic evolution of a CA that follows Rule 30 (also taken from [3]).

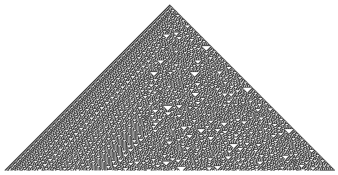

Fig. 2: Emergence of chaotic behavior with Rule 30

II. SAT encoding of the SPG problem

A. SPG as a SAT Benchmark

Given a population of $n \geq 1$ organisms, a maximum time step $T \geq 2$, , and a safety threshold $0 < P \leq 100$, the task of the SPG problem is to determine if the population evolve upto T by following Rule 30, with respect to the following two constraints:

safety: *Total number of alive organisms in every time step $1 \leq t \leq T$ is at least $n * (P/100)$.*

growth: *For any two consecutive time steps t and t', where $t' = t + 1$, number of alive cells at t' is greater or equals to the number of alive cells at t.*

We can encode an instance of the SPG problem as a SAT instance. Let s_i^t be the state of the current cell i, where $1 \leq t \leq T$ and $1 \leq i \leq n$. Given a SPG problem, we encode it as a SAT formula F_{SPG} as follows

$$F_{SPG} = F_{evolution} \cup F_{safety} \cup F_{growth} \cup F_{boundary}$$

, where, $F_{evolution}, F_{safety}, F_{growth}$, and $F_{boundary}$ are defined as follows:

$$F_{evolution} : \bigwedge_{t=1}^{T} \bigwedge_{i=1}^{n} (s_i^{t+1} = (s_{i-1}^t \oplus (s_i^t \vee s_{i+1}^t)))$$

$$F_{safety} : \bigwedge_{t=1}^{T} \sum_{i=1}^{n} s_i^t \geq n * (P/100)$$

$$F_{growth} : \bigwedge_{t=1}^{T-1} \sum_{i=1}^{n} s_i^{t+1} \geq \sum_{i=1}^{n} s_i^t$$

$$F_{boundary} : \bigwedge_{t=1}^{T} \neg s_0^t \wedge \neg s_{n+1}^t$$

Over T time steps,

- $F_{evolution}$ encodes the evolution of the population of n organisms that follows Rule 30.
- F_{safety} encodes the population safety constraint.
- F_{growth} encodes the population growth constraint.
- $F_{boundary}$ encodes the assertion that left neighbor (resp. right neighbor) of the leftmost (resp. rightmost) organism (resides outside of the *boundary* of a given population) of the population is always dead (0).

F_{SPG} is SATISFIABLE, if the population can evolve upto time step T with respect to the safety and growth constraint, otherwise, it is UNSATISFIABLE.

III. Problem Modeling and Instance Generation for the SPG benchmarks

A. Problem Modeling

picat [1] is a CSP solver, which accepts a CSP problem and converts it to a SAT CNF formula, which is inturn solved by a SAT solver hosted by **picat**. Before solving the converted CNF formula, **picat** outputs the CNF formula.

To generate instances for the SPG benchmark, we use this CNF generation feature of **picat**. First, we modelled the SPG problem in a **picat** program $picat_{SPG}$. Then, for a given set of parameter values for (T, n, P), we use this $picat_{SPG}$ model to generate CNF F_{SPG} by exploiting the CNF generation functionality of **picat**.

B. Instance Generation

We have generated a set of F_{SPG} instances with the $picat_{SPG}$ by varying the parameters T and n, while setting P to a fixed value of 70. From this set of instances, we have submitted 20 instances for SAT competition-2021 (CNF file names with prefix *spg*), 10 of which are interesting[1].

References

[1] Picat, http://picat-lang.org/resources.html, Accessed: 2020-04-09

[2] Stephen Wolfram, A new kind of science. Wolfram-Media 2002, ISBN 978-1-57955-008-0, pp. I-XIV, 1-1197.

[3] Rule 30 , https://mathworld.wolfram.com/Rule30.html, Accessed: 2020-04-09

[1]Not too easy (solvable by MiniSat in a minute) or too hard (unsolvable by the participants own solver within one hour on a computer similar to the nodes of the StarExec cluster)