



GROUP 3: Game Of Life

Emily Burke (22EB10)

Liam Ault (20LOJN)

Riche Cai (22TWQ4)

Martin Allerdissen (22SMJ3)

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

December 8, 2024

C1: Articulating the Problem

Abstract

Conway's Game of Life is a zero-player automated simulation, "played" on a grid of tiles.

note: Alive is a "True" tile with the boolean value 1 and dead is a "False" tile with the boolean value 0:

The grid size can be bound, or it can be infinite to simulate ever growing complexity. Each tile on the grid represents a cell, that can be alive or dead. The game starts with an initial configuration of the grid, and respectively the states of all cells on the grid.

With each iteration of the game, the state of a cell can change, determined by the rules of Conway's Game of life. Briefly, the number of alive neighboring cells determine the state during the subsequent iteration [see C2: Constraints].

Our model will explore configurations on an 8 x 8 grid. For an arbitrary tile, the 8 surrounding tiles are it's neighbors:

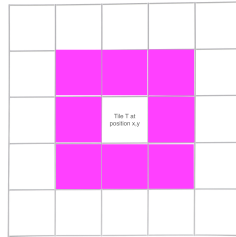


Figure 1: neighboring tiles for an arbitrary tile T , with coordinates x,y

C2: Modeling the Problem

Propositions

1. T_{xy} : represent a tile (cell) on an 8 x 8 grid:
 - Let T represent an alive tile, and the negation of T represent a dead tile.
 - Let xy represent a tiles coordinates i.e. tile location on the grid.
 - For any number, $n < 10$, of iterations each tile has a subsequent state determined by the rules of the game: the quantity of live neighbors i.e. how many of the 8 surrounding tiles are alive. Thus the state of a tile may or may not change for each iteration of the game.
2. C : grid configuration; the conjunction of all tile states on the grid:

-
- We introduce this as a proposition for simplicity as modeling the states of all the tiles would be difficult and confusing.
 - A grid configuration can be alive C or dead $\neg C$
 - Each configuration has an iteration number, in the documentation this is denoted by a subscript ($z = 0$ is the initial configuration), though it is modeled slightly different in the code. This is not necessarily important to the logical representation but is important to implementation of the model in the code.
3. **S**: stability: is true when there is no state change. Stability is true when a configuration exhibits a fixed state.
 4. **R**: repeating: is true when the state during some iteration matches that of a previous iteration.
 5. **G**: glider: is true when a configuration moves diagonally by one cell after 4 iterations
 6. **F**: fluctuation/ oscillation: is true when a grid evolves for a period of time, then returns to the initial configuration.

Constraints

We have limited the number of iterations to a maximum of 10 as to not overwhelm the SAT solver.

The constraints apply the rules of Conway's Game of Life to get the subsequent state of a tile, based on the number of alive neighbors:

- $T_{xy} \rightarrow T_{xy}$; an alive tile remains alive when it has 2 or 3 alive neighbors.
- $T_{xy} \rightarrow \neg T_{xy}$; an alive tile dies if it has less than 2 or more than 3 neighbors.
- $\neg T_{xy} \rightarrow T_{xy}$; a dead tile becomes alive when it has 3 alive neighbors.

We then use these rules to compile complete grid configurations on the 8 x 8 matrix.

- Configurations are dependent on the states of all the tiles on a grid because the configuration is equivalently the conjunction of all the tile states on the grid.

note: any combination of tiles represents a grid configuration, it is simply more legible to model an example rather than arbitrary tiles, the example below represents an initial configuration of a still block on the 8 x 8 grid:

$$(\neg T_{0,0} \wedge \dots \wedge \neg T_{4,2} \wedge T_{4,3} \wedge T_{4,4} \wedge \neg T_{4,5} \wedge \dots \wedge \neg T_{5,2} \wedge T_{5,3} \wedge T_{5,4} \wedge \neg T_{5,5} \wedge \dots \wedge \neg T_{8,8}) \iff C_0$$

note: the ellipses represent the intermediate dead tiles.

To avoid over-complication and ensure we have sufficient computational resources, it was simpler to build our model based on examples.

This means it cannot handle complex patterns, and only works for a subset of pre-defined states.

-
- We assert that a when a configuration is alive this implies that it is either stable or repeating or a glider.

$$C \rightarrow (S \vee F \vee G)$$

- Stability implies repetition, but repetition does not imply stability:

$$S \rightarrow R$$

$$\neg(R \rightarrow S)$$

or equivalently:

$$R \vee \neg S$$

- A grid configuration is dead if all the tiles on the grid are dead:

$$(\neg T_{0,0} \wedge \neg T_{0,1} \wedge \dots \wedge \neg T_{8,8}) \iff \neg C$$

- A dead configuration is stable and repeating:

$$\neg C \rightarrow (S \wedge R)$$

- If a living configuration has a repetition, then it will be stable or oscillating:

$$(C \wedge R) \rightarrow (S \vee F)$$

- If a configuration is not stable and has repetition, then it is fluctuating/oscillating:

$$(C \wedge (\neg S \wedge R)) \rightarrow F$$

- If a live configuration is not stable and not repeating, then it is a glider:

$$(C \wedge \neg S \wedge \neg R) \rightarrow G$$

C3: Model Exploration

Our implementation explores configurations on a 9 x 9 grid, which is represented in code as a matrix, where each entry is a tile. Breaking down the exploration, we are interested in the how a configuration changes throughout the game, and what it looks like after 10 iterations. Additionally, we explore possible configurations that oscillate:

- **Final configuration:** We can determine the final state given only the initial state, applying the rules of the game on each tile to get each subsequent state until the game is finished. In the original game, a configuration can return as dead, "still life" (we refer to this as stable or at times "inactive"), oscillating or a spaceship. For simplicity, the only spaceship we are interested in is a glider.

- Oscillating configurations:** exploring different periods of an oscillating grid. Without setting an initial state, our goal is to return some grid that has an oscillation period of 2 or 3.

Below are some examples of the possible configurations:

note: it is difficult to display an oscillating or glider configuration statically, however an oscillation is some configuration that changes during an iteration period, then returns to its initial state, while a glider is a configuration of some pattern, that moves diagonally by one tile after 4 iterations of the game.


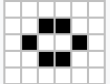
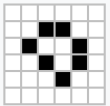


Still lifes	
Block	
Bee-hive	
Loaf	
Boat	
Tub	

Figure 2: Stable configurations




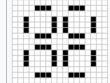
Oscillators	
Blinker (period 2)	
Toad (period 2)	
Beacon (period 2)	
Pulsar (period 3)	

Figure 3: Oscillating configurations

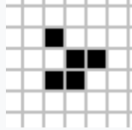
Spaceships	
Glider	

Figure 4: Glider Configuration

The code returns whether or not the test is satisfiable, how many solutions, and the value of every proposition once solved.

It also outputs a visual of the grid in viz.txt, and the individual values in the solution.txt.

Challenges

1. The first challenge we faced was how to correctly model the propositions and constraints in a manner that was concise, yet still logically sound. We tried to model the constraints in terms of tile states, however this quickly became overly complex and hard to follow. It became apparent that the best way to model our problem setting was by first modeling tile states (i.e. the rules of the game), then modeling how this related to an entire grid configuration. Ultimately this allowed us to show how entire configurations change throughout the game, which effectively simplified the model. In turn, the Jape proofs were far easier to construct. This also helped in terms of the code, as it was simpler to encode grid constraints as opposed to modeling individual tile constraints.
2. Further, we faced difficulty in terms of representing the iterations of the game. Initially we tried to model this within our constraints, however this would have significantly limited our model. Consequently, representation would not have allowed for dynamic constraints, and we faced this issue when trying to implement the code. Thanks to the help of our TA, this issue was brought to our attention and it became far easier to apply constraints. To elaborate slightly, when we were looking at repeating configurations, the constraint was modeled in terms of comparing iterations. The solution turned out to be as simple as portraying repetition generally, where this proposition evaluated to true when there was any instance of matching configurations. Although we did compare iterations in our code to determine repetition, this was not implemented as a constraint, allowing for better generalization.
3. Additionally, we wanted to try and explore more complex configurations and so it seemed most reasonable to recursively check tile states, applying the rules of the game to return the end configuration. We quickly realized this was not going to work because we needed to compare configurations and thus store them. Even though this simplified our model, it was really the only reasonable solution as comparisons were necessary to establish repetitions and oscillations. So, we tweaked the code to run each iteration

and store them all, then checking which constraints held up, allowing us to return configuration classifications.

D3: Jape Proofs

1: $C, C \rightarrow (S \vee F \vee G)$	premises
2: $G \rightarrow \neg R, F \rightarrow \neg R, S \rightarrow R$	premises
3: $\neg R \rightarrow \neg S$	Theorem $P \rightarrow Q \vdash \neg Q \rightarrow \neg P$ 2.3
4: $S \vee F \vee G$	\rightarrow elim 1.2,1.1
5: $S \vee F$	assumption
6: S	assumption
7: R	\rightarrow elim 2.3,6
8: $R \vee \neg S$	\vee intro 7
9: F	assumption
10: $\neg R$	\rightarrow elim 2.2,9
11: $\neg S$	\rightarrow elim 3,10
12: $R \vee \neg S$	\vee intro 11
13: $R \vee \neg S$	\vee elim 5,6–8,9–12
14: G	assumption
15: $\neg R$	\rightarrow elim 2.1,14
16: $\neg S$	\rightarrow elim 3,15
17: $R \vee \neg S$	\vee intro 16
18: $R \vee \neg S$	\vee elim 4,5–13,14–17
19: $C \wedge (R \vee \neg S)$	\wedge intro 1.1,18

Figure 5: Stability & Repetition Proof

The goal of this proof is to show that $\neg(R \rightarrow S)$ is valid within the context of our problem setting. We use the known properties of configurations and the fact that stability implies repetition to deduce the conclusion $C \wedge (R \vee \neg S)$.

In other words, if we are given a configuration C , that can be stable, fluctuating or a glider, the properties of such configurations, along with the fact that stability implies repetition, is it the case that a configuration is either repeating or not stable? Our proof confirms that there is some model where this is true.

This proof asserts that a glider configuration does not repeat:

1:	$\forall x.(C(x) \rightarrow S(x) \vee F(x) \vee G(x)), \forall x. \neg(S(x) \vee F(x))$	premises
2:	actual i	assumption
3:	$C(i) \rightarrow S(i) \vee F(i) \vee G(i)$	\vee elim 1.1,2
4:	$\neg(S(i) \vee F(i))$	\vee elim 1.2,2
5:	$C(i) \wedge \neg S(i) \wedge \neg R(i)$	assumption
6:	$C(i) \wedge \neg S(i)$	\wedge elim 5
7:	$C(i)$	\wedge elim 6
8:	$S(i) \vee F(i) \vee G(i)$	\rightarrow elim 3,7
9:	$S(i) \vee F(i)$	assumption
10:	$\neg G(i)$	assumption
11:	\perp	\neg elim 9,4
12:	$G(i)$	contra (classical) 10–11
13:	$G(i)$	assumption
14:	$G(i)$	\vee elim 8,9–12,13–13
15:	$C(i) \wedge \neg S(i) \wedge \neg R(i) \rightarrow G(i)$	\rightarrow intro 5–14
16:	$\forall x.(C(x) \wedge \neg S(x) \wedge \neg R(x) \rightarrow G(x))$	\forall intro 2–15

Figure 6: Glider Proof

We know that any configuration x can be on one of three types. Now, let's suppose we also know that a glider is not stable or fluctuating. This allows us to deduce that a glider does not have repetition because any configuration without the properties stability and repetition will be a glider.

1:	$(C \wedge R) \rightarrow (S \vee F)$	premise
2:	$C \wedge (\neg S \wedge R)$	assumption
3:	$\neg S \wedge R$	\wedge elim 2
4:	$\neg S$	\wedge elim 3
5:	R	\wedge elim 3
6:	C	\wedge elim 2
7:	$C \wedge R$	\wedge intro 6,5
8:	$S \vee F$	\rightarrow elim 1,7
9:	S	assumption
10:	\perp	\neg elim 9,4
11:	F	contra (constructive) 10
12:	F	assumption
13:	F	\vee elim 8,9-11,12-12
14:	$(C \wedge (\neg S \wedge R)) \rightarrow F$	\rightarrow intro 2-13

Figure 7: Oscillation Proof

The goal of this proof is to demonstrate that a living configuration that has repetition and is NOT stable must be oscillating. This proof begins with the statement that a living configuration that has repetition will either be stable or oscillating. Line 14 simply displays the scenario in which a configuration oscillates. We can see that in the conclusion, the configuration is alive and has repetition, which would satisfy the premise. However, because we specify that the configuration is NOT stable, then it must be oscillating.

D4: First-Order Extension

In order to extend out propositional model to predicate logic we can use the following predicates:

- $T(i)$: represents an alive tile
- $C(x)$: represents an alive configuration
- $S(x)$: represents a stable configuration
- $R(x)$: represents a repeating configuration

-
- $F(x)$: represents a fluctuating configuration
 - $G(x)$: represents a glider configuration

Here's how our constraints may change:

If all tiles are dead, then the configuration is dead:

$$\forall i \forall x (\neg T(i) \wedge S(x) \wedge R(x) \rightarrow \neg C(x))$$

All configurations are either stable, fluctuating or a glider:

$$\forall x C(x) \rightarrow \exists x (S(x) \vee F(x) \vee G(x))$$

If all configurations are repeating, then the configuration is stable:

$$\forall x (C(x) \wedge R(x)) \rightarrow \forall x S(x)$$

All configuration either have repetition or are not stable:

$$\forall x C(x) \rightarrow \exists x (R(x) \vee \neg S(x))$$

All configurations that return either stable or fluctuating implies there is a configuration with the property repeating:

$$\forall x (C(x) \wedge (S(x) \vee F(x))) \rightarrow \exists x R(x)$$

If a configuration returns fluctuating, then there is some configuration that has properties not stable, and repeating:

$$\exists x (C(x) \wedge F(x)) \rightarrow \exists x (\neg S(x) \wedge R(x))$$

For some configuration that returns a glider, there is some configuration that is not stable and not repeating:

$$\exists x (C(x) \wedge G(x)) \rightarrow \exists x (\neg S(x) \wedge \neg R(x))$$