Compositional Object Synthesis in Game of Life Cellular Automata Using SAT Solver

Haruki Nishimura and Koji Hasebe

Department of Computer Science, University of Tsukuba
1-1-1, Tennodai, Tsukuba 305-8573, Japan
nishimura@mas.cs.tsukuba.ac.jp, hasebe@cs.tsukuba.ac.jp

Abstract. Conway's Game of Life is a two-dimensional cellular automata known for the emergence of objects (i.e., patterns with special properties) from simple transition rules. So far, various interesting objects named still-life, oscillator, and spaceship have been discovered, and many methods to systematically search for such objects have been proposed. Most existing methods for finding objects have comprehensively search all patterns. However, attempting to obtain a large object in this way may cause a state explosion. To tackle this problem and enhance scalability, in this study, we propose a method to generate objects by synthesizing some existing objects. The basic idea is to arrange multiple pieces of existing objects and compose them by complementing the appropriate patterns. The problem of finding complementary patterns is reduced to the propositional satisfiability problem and solved using SAT solver. Our method can reduce the object generation time compared to the case where a large object is generated from the beginning. We also demonstrate the usefulness of our proposed method with an implementation for automatic object generation.

Keywords: Cellular Automata, Game of Life, SAT Solver, Object Synthesis

1 Introduction

Conway's Game of Life [19] is a two-dimensional cellular automata known for the appearance of objects (i.e., patterns with special properties). Objects in Game of Life have attracted significant attention due to their behavior that resembles living things. So far, various interesting objects named still-life, oscillator, and spaceship have been discovered, and many methods to systematically search for such objects have been proposed. Harold [16] proposed a method using De Bruijn's table [9], which was partially implemented as an object generation tool [10]. Knuth mentioned in his book [13] how to reduce the object generation problem to the satisfiability problem then showed that the problem can be solved using a SAT solver. These methods can comprehensively search for all objects that fall within a specific rectangular range. However, it is basically a brute force search for patterns that satisfy the conditions for becoming an object, thus there is a problem that the larger the size of the object, the longer it takes to search.

To avoid this scalability issue, a possible approach is to reuse the existing objects to reduce the computational cost, that has not been thoroughly investigated.

In this study, we propose a method to generate a large object by synthesizing multiple existing objects in Game of Life. The basic idea is to create a new object (called a chimera) by joining a fragment of existing object with a fragment of another object. The problem of finding complementary patterns is reduced to the propositional satisfiability problem and solved using SAT solver.

We implement an automatic object synthesis tool using Python based on the method described above. Using Z3py [1] as the SAT solver, we obtained that new objects can be created by synthesizing objects belonging to the same category for each of the three categories of still-lifes, oscillators, and spaceships. In our experiments, we observed that new oscillator was obtained from two oscillators with different periods.

The remainder of the paper is organized as follows. Section 2 presents an overview of the Game of Life. Section 3 describes the method for synthesizing objects using the SAT solver. In Section 4, we present some examples of synthesizing objects. Section 5 describes related work. Finally, Section 6 concludes the paper and presents future work.

2 Overview of the Game of Life

This section briefly presents an overview of the Game of Life and defines some related concepts.

2.1 Rules

A cellular automaton is an automaton in which multiple cells spread over a grid space change their state according to the state of neighboring cells. In Game of Life, cells in the state of 0 or 1 are spread in a two-dimensional lattice space, and transition with time according to the following three rules.

Birth: If the state of a cell is 0 and the state of exactly three cells out of the touching eight cells (called neighbors) is 1, then its state transitions to 1 at the next time.

Survival: If the state of a cell is 1 and the state of two or three cells in the neighbors is 1, then its state transitions to 1 at the next time.

Death: A cell that does not meet either of the above two rules will transition to 0 at the next time.

The formal definition of the Game of Life can be given as follows.

Definition 1 (Game of Life). Let $\mathbb{N}$ be the set of natural numbers. For a two-dimensional lattice space $P \subseteq \mathbb{N} \times \mathbb{N}$, the cell at the intersection point of i-th row (from the top) and j-th column (from the left) is denoted by (i, j). Let the time $t = 0, 1, 2, \dots$ ($\in T = \mathbb{N}$). The state (0 or 1) of the cell (i, j) at a certain time t is defined as the function $\sigma : P \times T \to \{0, 1\}$, and its value is denoted by

$x^t_{i,j}$. A Game of Life with area P is a pair of σ (called the rules of the Game of Life) and the area P_{GoL} satisfying the following formula for any $(i,j) \in P$ and $t \in T$.

$$\sigma(x^t_{i,j}) = \begin{cases} 1 \ (\mathit{if} \sum_{m=i-1}^{i+1} \sum_{n=j-1}^{j+1} x^t_{m,n} = 3) \\ 1 \ (\mathit{if}\, x^t_{i,j} = 1 \ \mathit{and} \sum_{m=i-1}^{i+1} \sum_{n=j-1}^{j+1} x^t_{m,n} = 2) \\ 0 \quad \mathit{otherwise.} \end{cases}$$

2.2 Boolean Representation of Rules

By considering the state of a cell as a Boolean value such that 1 is "true" and 0 is "false," function σ representing the rules of Game of Life can be regarded as a Boolean function δ that takes nine states of a cell and its neighbors as arguments and returns its state at the next time step. Based on the above consideration, we define $\delta(V(x^t_{i,j})) = \sigma(x^t_{i,j})$, where $V(x^t_{i,j}) = (x^t_{i-1,j-1}, x^t_{i-1,j}, x^t_{i-,j+1}, x^t_{i,j-1}, x^t_{i,j}, x^t_{i,j+1}, x^t_{i+1,j-1}, x^t_{i+1,j}, x^t_{i+1,j+1})$.

2.3 Objects

There are patterns with special properties in Game of Life. In this study, such a pattern is called an object. Objects can be classified into several categories according to their properties, and the following three are well known.

Still-life: where the shape does not change, no matter how many transitions are made;
Oscillator: that returns to the original shape after p transitions;
Spaceship: that returns to the original shape while moving in a certain direction through p transitions are known.

Formally, these categories of the objects can be defined as follows.

Definition 2 (Object). Rectangular area $R \subseteq P_{GoL}$ is called a spaceship with the direction (a,b) and period p if $x^{t+p}_{i,j} = x^t_{i-a,j-b}$ (i.e., $\sigma^p(x^t_{i,j}) = x^t_{i-a,j-b}$) holds for any $t \in T$ and $(i,j) \in R$. A spaceship with direction $(0,0)$ is called an oscillator with a period p, and an oscillator with period 0 is called a still-life.

3 Object Synthesis

3.1 Definition

Object synthesis proposed in this paper is based on the method of creating a new object by taking out fragments of two existing objects with the same period and direction and inserting an appropriate pattern between them. Here, the synthesis of two fragments with different periods is possible by taking common multiples for both periods. For the new pattern created by synthesis to become an object with period p, it is required that the states of all cells in the fragments and its surrounding within p do not change after the transition of σ^p (i.e., applying σ p times). Our object synthesis is defined as follows. We consider the case where the two objects are arranged side by side and combined. However, the same definition can be made when two objects are arranged vertically.

Definition 3 (Object synthesis). Let $m_{max}, m_{min}, n_{max}, n_{min}, mid_u, mid_v, a, b, p \in \mathbb{N}$ be some specific values with $mid_u < mid_v$. Let $R_0, R_1 \subseteq P_{GoL}$ be rectangular areas. For R_0, R_1, we define rectangular areas $R_l(\subseteq R_0)$ and $R_r(\subseteq R_1)$ as follows:

– $R_l = \{(i,j) \mid m_{min} \le i \le m_{max},\ n_{min} \le j \le mid_u\}$.
– $R_r = \{(i,j) \mid m_{min} \le i \le m_{max},\ mid_v \le j \le n_{max}\}$.

A rectangular area $R_2 = \{(i,j) \mid m_{min} \le i \le m_{max},\ n_{min} \le j \le n_{max}\}$ is called a synthesized object from R_l and R_r if the following formula

$$\sigma^p(x_{i,j}) \leftrightarrow x_{(i-a),(j-b)} \tag{1}$$

is true for all $(i,j) \in R_s = \{(i,j) \mid m_{min} - p \le i \le m_{max} + p,\ n_{min} - p \le j \le n_{max} + p\}$.

Here, R_l and R_r are called left-side and right-side fragments, respectively. Besides, $C_l = \{(i, mid_u) \mid m_{min} \le i \le m_{max}\}$ and $C_r = \{(i, mid_v) \mid m_{min} \le i \le m_{max}\}$ are called the joint edges of R_l and R_r, respectively. Rectangular area $R_b = \{(i,j) \mid m_{min} \le i \le m_{max},\ mid_u + 1 \le j \le mid_v - 1\}(\not\subseteq R_0, R_1)$ is called a complementary pattern of R_l and R_r.

In Definition 3, the formula (1) must hold for all cells in R_2 and its surrounding within p. To obtain a synthesized object, it is sufficient to choose a pattern of R_b so that the formula (1) holds for only cells in R_b and its surrounding within p. This is because R_l and R_r are parts of the existing objects. Thus, the cells in R_l or R_r and in the area outside R_b and its surrounding within p should satisfy the formula. In this study, we call R_m the area R_b and its surrounding within p. If there exists a pattern such that all cells in R_b satisfy the following formula, then, we obtain a synthesized object.

$$\bigwedge \sigma^p(x_{i,j}) \leftrightarrow x_{i-a,j-b} \quad (x_{i,j} \in R_m). \tag{2}$$

The problem of determining whether there exists an assignment of Boolean values to each variable that satisfies a given propositional formula is known as the Boolean satisfiability problem (SAT) [6].

Although this problem is known to be NP-complete, for the formulas in conjunctive normal form (CNF), Davis-Putnam-Logemann-Loveland (DPLL) algorithm [8] and the conflict-driven clause learning [14, 3] have been discovered to reduce the search space. However, various SAT solver based on these algorithms have been developed. The idea of our object synthesis is to derive the complementary pattern by converting the above formula into CNF and solve it using a SAT solver.

3.2 Complementary Pattern

The definition of the complementary pattern in the object synthesis described above is given as follows.

Definition 4 (Complementary Pattern). Let $e_x, e_y, p, a, b \in \mathbb{N}$ be specific values. Let R_e be a spaceship (with period p and direction (a, b)) defined as follows.

$$R_e = \{(i, j) \mid e_y - p \leq i \leq e_y + m + p - 1,\ e_x - p \leq j \leq e_x + n + p - 1,\ m > 0,\ n > 0\}.$$

A rectangular area $R_c = \{(i, j) \mid e_y \leq i \leq e_y + m - 1,\ e_x \leq j \leq e_x + n - 1,\ m > 0,\ n > 0\}$ $(\subseteq R_e)$ is called a complementary pattern of R_e with period p and direction (a, b) if $\sigma^p(x^t_{i,j}) = x^t_{i-a,j-b}(-p \leq a, b \leq p)$ is satisfied for all $(i, j) \in R_e$.

As seen in Definition 2, the complementary patterns for oscillator and still-life can be obtained by considering the case that $(a, b) = (0, 0)$ and the case that both $(a, b) = (0, 0)$ and $p = 0$, respectively.

Since the formula δ^p is equivalent to σ, by Definition 4, the complementary pattern R_c satisfies the following expression.

$$\bigwedge_{i=e_y-p}^{e_y+m+p-1} \bigwedge_{j=e_x-p}^{e_x+n+p-1} (\delta^p(V(x^t_{i,j})) \leftrightarrow x^t_{i-a,j-b}) \tag{3}$$

where $V(x^t_{i,j}) = \{x^t_{i+u,j+v} \mid -p \leq u, v \leq p\}$. By this translation, the problem of object synthesis can be reduced to a satisfiability problem. Formula (3) is called the complementary pattern constraints. The expression $\delta^p(V(x^t_{i,j})) \leftrightarrow x^t_{i-a,j-b}$ in Formula (3) is called the constraints on cells in the object.

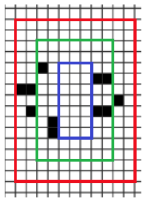

Fig. 1. Example of synthesizing an oscillator with period 2.

Example 1. As a concrete example of the complementary pattern, we present the complementary pattern for synthesizing oscillators. Let us consider the procedure for combining the two oscillators shown in Figure 1 into one oscillator by filling the blue frame with a complementary pattern. To generate such a complementary pattern, first, cell constraints are created for the 8 $\times 4$ cells shown in the green frame. The complementary pattern constraints are expressed by connecting them with logical product. Since the state of one cell around it is required to create a constraint for one cell, 10 $\times 6$ cells in the red frame are referenced. The state of each cell inside the blue frame is represented as a variable, while that of the outside is represented as a constant. If the cell at the upper left corner of the red area is called $(1, 1)$, We express the constraint of the completion pattern as follows.

$$\bigwedge_{i=2}^{9} \bigwedge_{j=2}^{5} (\delta(V(x^t_{i,j})) \leftrightarrow x^t_{i,j}). \tag{4}$$

Here, the cell constraints for $(i, j) = (2, 5)$ and $(i, j) = (6, 4)$ are, respectively, represented by the following formulas.

- $0 \leftrightarrow \delta(0, 0, 0, 0, 0, 0, x^t_{3,4}, 1, 1)$.
- $x^t_{6,4} \leftrightarrow \delta(x^t_{5,3}, x^t_{5,4}, 1, x^t_{6,3}, x^t_{6,4}, 0, x^t_{7,3}, x^t_{7,4}, 1)$.

A new object can be obtained by joining the fragments taken out from the two existing objects with a complementary pattern. We call it a chimera. The following proposition guarantees that a chimera obtained from the two objects with period p and direction (a, b) is actually an object with the same period and direction.

Proposition 1. *Let R ($\subseteq P_{GoL}$) be a rectangular area consisting of three rectangular areas R_0, R_1, R_2. Suppose $\sigma^p(x^t_{i,j}) = x^t_{i-a,j-b}(-p \le a, b \le p)$ holds for all $(i, j) \in R_0 \cup R_1$, and $\sigma^p(x^t_{i,j}) = x^t_{i-a,j-b}$ holds for all $(i, j) \in R_2$, then R is an object with period p and direction (a, b).*

The two fragments and their complementary patterns correspond to R_0, R_1, and R_2 in the above theorem, respectively. From Definition 4, the complementary pattern does not change the boundary pattern between R_0 and R_2 as well as between R_1 and R_2. Thus, $\sigma^p(x^t_{i,j}) = x^t_{i-a,j-b}(-p \le a, b \le p)$ holds for all $(i, j) \in R_0 \cup R_1$. From the same definition, the expression $\sigma^p(x^t_{i,j}) = x^t_{i-a,j-b}(-p \le a, b \le p)$ holds for all (i, j) in the complementary pattern. Thus, $\sigma^p(x^t_{i,j}) = x^t_{i-a,j-b}(-p \le a, b \le p)$ holds for all $(i, j) \in R_2$. From the above theorem, the chimera is an object with period p and direction (a, b).

In closing this section, we would like to note a limitation on our object synthesis. As mentioned in Definition 3, when synthesizing an object, a fragment taken from an existing object must always share at least three sides with the original object. For example, the fragment of the object shown in the blue frame in Figure 2 must be cut out so as to share the top, bottom, and left sides with the original object, as in the patterns surrounded by the red frames in (a) and (b). Therefore, the pattern shown by the red frame in (c) is not a fragment of the original object. The fragment may be the same as the original object. However, in that case, it is allowed that the complementary pattern becomes a pattern composed of all 0s. In order to avoid such trivial composition, it is necessary to add some new constraints to the complementary pattern.

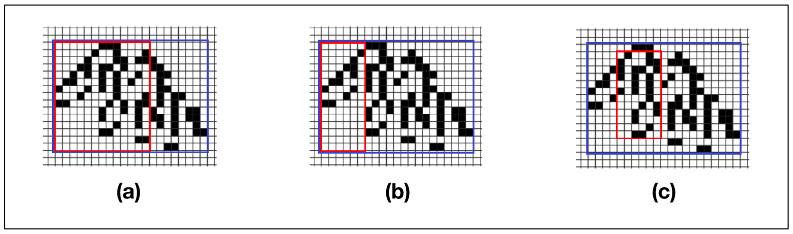

Fig. 2. Example of correct/incorrect fragments of an object.

4 Implementation

We have implemented a simple tool for automating the object synthesis method described above. Here, the Python library Z3py is used to derive the propositional function δp and cell constraints. Besides, this library is used to derive variable assignments, making it possible to satisfy the constraints of the completion pattern.

Below are some examples of objects produced by synthesis using our implementation. Figure 3 shows an example of synthesizing an oscillator. Here, the objects (a), (b), and (c) are oscillator with period of 2, fixed object, and oscillator with a period of 3, respectively. First, we picked a fragment from these three objects (as shown by the red frames in (d) of the figure). Next we synthesized fragments of (a) and (b) by inserting a complementary pattern (as shown by the left blue frame in (d)) and further synthesized the pattern obtained thereby and the fragment of (c) by inserting a complementary pattern (as shown by the right blue frame). As a result, we obtained an oscillator with period of 6, as shown in (d) was finally obtained. The times required for these syntheses were about 60 and 290 seconds, respectively.

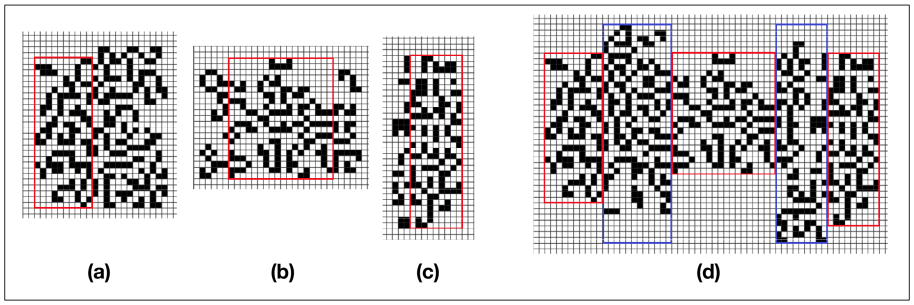

Fig. 3. Example of synthesizing an oscillator.

Figure 4 shows an example of synthesizing a spaceship. In this figure, both (a) and (b) are spaceships with period 2 and direction $(0, -1)$, respectively. Similar to the previous example, we first picked a fragment (as shown by the red frames in (c) in the figure) from each of the objects, respectively. By inserting a complementary pattern (as shown by the blue frame) between these patterns, we obtained a spaceship with period 2 and direction $(0, -1)$.

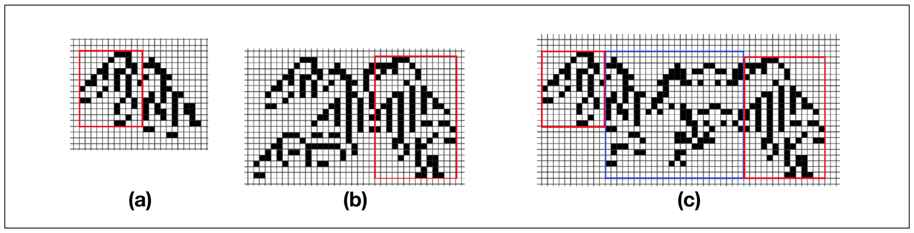

Fig. 4. Example of synthesizing a spaceship.

As shown in the above examples, we demonstrated that our method correctly performed the synthesis. In the oscillator's synthesis, it was shown that oscillators with different periods can be synthesized. It is suggested that an oscillator with a larger period can be generated by repeating this operation. From the previous examples, the size of the complementary pattern for synthesis is generally as large as or larger than each fragment to be synthesized, so it is necessary to keep sufficient space between the fragments for synthesis. If the search for the complementary pattern fails due to insufficient space, it is necessary to redo the search at a wider space. Thus, finding the minimum size of the complementary pattern required for synthesis is useful for shortening the search time. However, this is unclear and further investigation is required. To reduce the time to convert the constraint of each cell to CNF, efficient conversion technique, such as Tseitin conversion would be useful.

5 Related Work

De Bruijn diagram [9] has been widely used as a useful tool in the study of cellular automata. McIntosh [15, 17] proposed a method to enumerate still-life by arranging predetermined small patterns. The basic idea was that the patterns of 3×3 square appearing in still-life were classified into 284 types of patterns, then still-lifes of any size and patterns were generated by connecting 3×3 square patterns in the catalog one-by-one according to some connection rules. First, $3 \times n$ patterns were generated by 3×3 square belonging to the catalog is generated as a horizontally long pattern of $3 \times n$ based on the connection rules of two 3×2

rectangles. Next, $m \times n$ pattern is completed by stacking the previously obtained $3 \times n$ patterns while shifting them vertically one cell at a time.

A tool for creating an object using the De Bruijn diagram is Eppstein's gfind [10, 11]. This tool used Dr Bruijn diagram to search for a spaceship, which is a kind of gridder.

Bounded model checking [4] is a model checking [5], which limits the transition sequences of a state transition system to a finite length for verification. In [13], Knuth suggested how to apply the logical formula used in the bounded model checking to convert the problem of object generation in Game of Life into satisfiability problem. Let $T(X, X')$ be a logical formula that returns true if the transition from pattern X is reachable to X'. Here, if X_n is reachable from X_0 by n-times state transitions, the formula $\bigwedge_{k=1}^{n} T(X_{k-1}, X_k)$ is true. If $X_0 = X_n$, then this formula returns true when X_0 is an oscillator with period n. Thus, if this equation is converted to CNF and solved using the SAT solver, the concrete pattern of the oscillator can be obtained. By adding some suitable conditions and changing values of the parameters, we can derive still-life, spaceship, and garden of Eden. In [13], it was introduced how to convert the expression $T(X, X')$ to a CNF, which facilitates the solution using the SAT solver. There have been some implementations of this method, such as Cunningham's Logic Life Search [7] and Goucher's ikpx [12].

6 Conclusions and Future Work

In this study, we proposed a method for synthesizing objects in Conway's Game of Life. The basic idea is to join two existing objects by inserting an appropriate pattern to obtain a new object. The constraints that the complementary pattern between the fragments must satisfy is described as a logical formula in which the state of each cell constituting the pattern is a variable. This formula becomes true if and only if a pattern that satisfies the constraint is input. Thus, the object synthesis problem can be reduced to the satisfiability problem, which can be solved using the SAT solver. We implemented the proposed method using Python with Z3py as an SAT solver and demonstrated the synthesis of various objects, such as still-life, oscillator, and spaceship.

Future research will consider finding regularity in shape from a large number of automatically generated objects by synthesis. Objects with the same period and direction of movement partially share some specific patterns while others are composed of different patterns. From this observation, it is expected that various objects can be generated by applying a (set of) certain non-deterministic manipulation rules to a specific pattern. We especially want to clarify such rules by focusing on the relationship with stochastic one-dimensional cellular automata.

References

1. z3py-tutorial, https://github.com/ericpony/z3py-tutorial.
2. A. Adamatzky. *Game of Life Cellular Automata*, Springer, 2010.

3. R. J. Bayardo Jr. and R. C. Schrag. Using CSP look-back techniques to solve real world SAT instances, *14th AAAI*, pp.203–208, 1997.
4. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, Bounded model checking, *Advances in Computers*, vol.58, pp.118-149, 2003.
5. E. Clarke, O. Grumberg, D. Kroening, D. Peled, H. Veith. *Model Checking (2nd Edition)*, MIT press, 2018.
6. S. A. Cook. The Complexity of Theorem-Proving Procedures, *3rd Annual ACM Symposium on Theory of Computing*, pp.151–158, 1971.
7. O. Cunningham. logic-life-search, https://github.com/OscarCunningham/logic-life-search.
8. M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving, *Communications of the ACM,* vol.5(4), pp.394-397, 1962.
9. N. G. De Bruijn. A combinatorial problem, *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, vol.49, pp.758-764, 1946.
10. D. Eppstein. gfind 4.9 – search for gliders in semitotalistic cellular automata, https://www.ics.uci.edu/ eppstein/ca/gfind.c.
11. D. Eppstein. Searching for Spaceships, *arXiv prepring*, cs/0004003, 2000.
12. A. P. Goucher. Metasat, https://gitlab.com/apgoucher/metasat.
13. D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*, Addison-Wesley Professional, 2015.
14. J. P. Marques-Silva and K. A. Sakallah. GRASP: A New Search Algorithm for Satisfiability, *Digest of ICCAD*, pp.220–227, 1996.
15. H. V. McIntosh. Linear Cellular Automata via de Bruijn Diagram, Universidad Autónoma de Puebla, 1991.
16. H. V. McIntosh. Life's still lifes, *Game of Life Cellular Automata*, pp.35-50, 2010.
17. H. V. McIntosh. Commentaries on The Global Dynamics of Cellular Automata, http://delta.cs.cinvestav.mx/ mcintosh/oldweb/wandl/wandl.html.
18. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver, *38th Annual Design Automation Conference*, pp.530-535, 2001.
19. J. L. Schiff. *Cellular Automata: A Discrete View of the World*, John Wiley & Sons, Inc., 2011.