SAT-based Analysis of Cellular Automata

Massimo D'Antonio and Giorgio Delzanno

Dip. di Informatica e Scienze dell'Informazione - Università di Genova via Dodecaneso 35, 16146 Genova, Italy, e-mail: giorgio@disi.unige.it

Abstract. Cellular Automata are a powerful formal model for describing physical and computational processes. Qualitative analysis of Cellular Automata is in general a hard problem. In this paper we will investigate the applicability of modern SAT solvers to this problem. For this purpose we will define an encoding of reachability problems for Cellular Automata into SAT. The encoding is built in a modular way and can be used to test *inverse reachability* problems in a natural way. In the paper we will present experimental results obtained using the SAT-solver zChaff.

1 Introduction

Cellular Automata (CAs) [18] are decentralized spatial extended systems consisting of large numbers of simple identical components with local connectivity. Such systems have the potential to perform complex computations with a high degree of efficiency and robustness, as well as to model the behavior of complex systems in nature. For these reasons CAs have been studied extensively in natural sciences, mathematics, and in computer science. For instance, they have been used as parallel computing devices for image processing, and as abstract models for studying cooperative or collective behavior in complex systems (see e.g. [2,3,21,11]).

Quantitative vs Qualitative Analysis Several tools have been used to animate specifications of CAs and to perform statistical analysis of their behavior, e.g., for car traffic or artificial life processes simulation (for a survey see e.g. [22]). Simulation amounts to compute all possible reachable configurations. For deterministic CAs this operation is fairly simple but some care must be taken in the way the transition rule is stored.

Differently from plain simulation, a *qualitative analysis* of the behavior of a CA can be a difficult problem to solve. In fact, problems like reachability of subconfigurations or existence of a predecessor configuration for generic CAs have been shown to be exponentially hard [6,19,20]. The hardness of some computational problems for CAs has been exploited for interesting applications. As an example, reversibility (undecidable in general [10]) has been used for designing cryptographic systems based on CAs [9].

Towards a Practical Solution of Hard Problems In recent years practical solutions to large instances of known hard problems like SAT (satisfiability of propositional formulas, a well known NP-complete problem) have been made possible by the application of specialized search algorithms and pruning heuristics. The connection between the complexity of some interesting problems for CAs and SAT (see e.g. [6]) suggests us a possible new application of all these technologies.

Technical Contribution In this paper we will investigate in fact the applicability of SAT solvers to the *qualitative analysis* of CAs. Specifically, following the *Bounded Model Checking* (BMC) philosophy introduced in [1], we will define a *polynomial time* encoding of a *bounded number* of evolution steps of a CA into a formula in *propositional logic*. Our encoding allows us to specify in a *declarative* and *modular way* several decision problems for CAs like (inverse) reachability as a SAT problem. As a result, we obtain an effective verification procedure for the analysis of CAs by resorting to efficient-in-practice existing SAT solvers like [7,8,16,14].

As preliminary experiments, we have selected a non-trivial example of CA, namely Mazoyer's solution to the Firing Squad Synchronization Problem [12]. In this example the length of the evolution (the diameter of the model in the terminology of BMC) leading to a successful final configuration depends on the dimension of the cellular space. For this reason, this problem is adequate to check qualitative problems that can be encoded as *bounded reachability*, e.g., checking if the final solution is the correct one, computing predecessor configurations, computing alternative initial configurations leading to the same solution, etc. We will use this case-study to test the performance of one of the fastest existing SAT-solvers called zChaff [14]. zChaff [14] can handle problems with up to 10^6 propositional variables. In our setting zChaff returns interesting results for problems of reasonable size (e.g. cellular spaces with 70 cells and evolution of 140 steps). Some built-in heuristics of zChaff however turned out to be inadequate for CA-problems like inverse reachability. We believe that specialization of SAT-solving algorithms to problems formulated on CAs could be an interesting future direction of research.

Plan of the Paper In Section 2 we introduce the notion of Cellular Automata. In Section 3 we define the encoding of the evolution of a CA and of its qualitative problems into a SAT formula. In Section 4 we encode several interesting properties of CAs. In Section 5 we discuss experimental results obtained with existing solvers. In Section 6 we discuss related works and future directions of this research.

2 Cellular Automata (CAs)

A CA consists of two components. The first component is a *cellular space*, i.e. a collection of identical finite-state machines (cells), each with an identical pattern

of local connection to other cells. Let S be the set of states of each automata. Each cell is denoted by an index $\boldsymbol{i}$. The *neighborhood* of a cell i is the set of cells of the network which will locally determine the evolution of i.

The second component is a transition rule δ that gives the update state for each cell $\boldsymbol{i}$ in function of the state of the cells in its neighborhood. In a CA a *discrete global clock* provides an update signal for all cells: at each time step all cells update their states synchronously according to δ.

Formally, we have the following definition.

Definition 1. *A d-CA $\mathcal{A}$ is a tuple $\langle d, S, N, \delta\rangle$ where*

- *$d \in \mathbb{N}$ is the dimension of the cellular space ($\mathbb{Z}^d$);*
- *S is the finite set of states of $\mathcal{A}$;*
- *$N \subseteq \mathbb{Z}^d$ is the* neighborhood *of $\mathcal{A}$ and $\mid N \mid = n$;*
- *$\delta : \ S^{n+1} \rightarrow S$ is the transition rule of $\mathcal{A}$.*

The meaning of the neighborhood of $\mathcal{A}$ is as follows. Suppose that $N = \{\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}\}$. Given a cell $\boldsymbol{i}$, its neighborhood is obtained then by considering the cells $\boldsymbol{i}, \boldsymbol{i} + \boldsymbol{v_1}, \ldots, \boldsymbol{i} + \boldsymbol{v_n}\}$. A classical examples is von Neumann's neighborhood defined as $N_N = \{(1,0),(0,1),(-1,0),(0,-1)\}$ for 2-CA.

2.1 Evolution of a CA

The computational meaning of a d-CA $\mathcal{A} = \langle d, S, N, \delta\rangle$ with $N = \{\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}\}$ is defined as follows.

A *configuration* of a $\mathcal{A}$ is defined as a function $c : \mathbb{Z}^d \rightarrow S$ that assigns a state to each cell of the cellular space. We will use $\mathcal{C}$ to denote the set of configurations.

The *global evolution function* $G_{\mathcal{A}}$ associated to $\mathcal{A}$ is a transformation from configurations to configurations such that

$$G_{\mathcal{A}}(c)(\boldsymbol{i}) = \delta(\langle c(\boldsymbol{i}), c(\boldsymbol{i} + \boldsymbol{v}_1), ..., c(\boldsymbol{i} + \boldsymbol{v}_n)\rangle) \text{ for any } c \in \mathcal{C}, \ \boldsymbol{i} \in \mathbb{Z}^d.$$

Given an initial configuration c_0, the evolution of $\mathcal{A}$, written $Ev^{\mathcal{A}}(c_0)$, is a sequence $\{c_t\}_{t\geq 0}$ of configurations such that $c_{t+1} = G_{\mathcal{A}}(c_t)$ for any $t \geq 0$. A configuration c' is *reachable* in k steps from configuration c_0 if there exists an evolution $\{c_t\}_{t\geq 0}$ such that $c' = c_k$. A configuration c' is *reachable* from c_0 if it is reachable in $k \geq 0$ steps.

Previous definitions are for infinite cellular spaces but computer simulations are obviously constrained on finite spaces, hence periodic boundary conditions (for example, the cellular space is defined as a ring) or the definition of constant boundary cells are necessary.

Example 1. Fig. 1 illustrates an example of local rule defined over $S = \{0, 1\}$ and $N = \{-1, 1\}$, given in a tabular form, and of an evolution on a circular cellular space. In the table representing the local rule the two columns C^t and C^{t+1} denote the state of a cell at time t and $t+1$ respectively, whereas columns N_1^t and N_2^t denote the state of cells in the neighborhood. In the evolution we highlight the neighborhood to which a rule is applied.

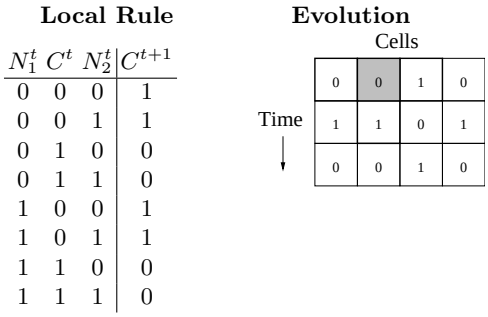

Fig. 1. A simple CA.

2.2 Expressiveness and Computational Complexity

CAs are a powerful computational model. In fact, it is sufficient to consider one-dimensional CAs to simulate Turing Machines. Given the richness of the model, several decision problems related to the computational interpretation of the evolution of CAs are hard or impossible to solve. The hardest problems are often related to the inverse exploration of the configuration space of a CA. Note, in fact, that the *past history* of a configuration contains non-deterministic choices for its predecessors. For instance, let us consider the Predecessor Existence Problem, defined as follows:

(PEP) Given a d-CA $\mathcal{A}$ *and* $x \in \mathcal{C}$, $\exists y \in \mathcal{C}$ *such that* $x = G_{\mathcal{A}}(y)$*?*

PEP is in NP for "finite" d-CAs, NP-complete for $d > 1$, and NLOG for 1-CAs. Proofs of NP-completeness of this kind of problems are often given via reductions of 3SAT into the decision problem taken into consideration. In this paper we will try to exploit reductions going in the opposite direction, namely from CA to SAT, in order to obtain an effective and flexible method for the analysis of difficult decision problems for CAs.

3 From Cellular Automata to Propositional Logic

In this section we will define a representation of a finite prefix of the evolution of a CA in propositional logic. The resulting formula will be used later to define several interesting decision problems for CAs in a formal and modular way. Furthermore, it represents the first step towards the use of SAT solvers for the analysis of CAs.

Symbolic Representation of Configurations Given a CA $\mathcal{A}$, we first represent its set of states $S = \{s_1, \ldots, s_n\}$ using a binary encoding over $m = \lceil log_2 n \rceil$ bits of a given choice of ordering numbers. Let us call $S' = \{w_1, \ldots, w_n\}$ be the resulting set of binary representations, where each $w_i \in \{0,1\}^m$ (sequence

of m bits). As an example, for $S = \{red, blue, yellow\}$ we can use 2 bits and the following encoding $w_{red} = 00$, $w_{blue} = 01$, and $w_{yellow} = 10$. Every state represented in binary can be naturally encoded as a propositional formula as follows. Let ℓ be a label (i.e. a string used later for stamping predicate symbols with time and position) and w a sequence of m bits $b_1 \dots b_m$. Furthermore, let . be an operator for concatenating labels. Then, we define the formula encoding b as follows:

$$Cod_w(w, \ell) \doteq \bigwedge_{i=1}^{m} Cod_b(b_i, i.\ell) \text{ where } Cod_b(b, \ell) \doteq \begin{cases} x_\ell \ if \ b = 1 \\ \neg x_\ell \ \ if \ b = 0 \end{cases}$$

Going back to our example, assume that 0.0 represents a cell in position 0 at time 0, then $Cod_w(w_{blue}, 0.0) = \neg x_{1.0.0} \wedge x_{2.0.0}$. Generalizing this idea, in the following we will use strings of the form $p.t$, where p is denotes a position and t a time-stamp, as labels for encoding the evolution of configurations using propositional formulas.

Specifically, let us assume that the cellular space is linearized into the range $1, \dots, N$. Then, a configuration is simply a tuple $c = \langle\langle 1, w_1\rangle, \dots, \langle N, w_N\rangle\rangle$, where w_i is the binary representation of a state of $\mathcal{A}$.

The encoding of c at instant t is defined then as follows:

$$Cod_c(c, t) \doteq Cod_w(w_1, 1.t) \wedge \dots \wedge Cod_w(w_N, N.t)$$

Thus, $Cod_c(c, t)$ gives rise to a conjunction of *literals* over the set of predicate symbols $x_{b.p.t}$ where $b \in \{1, \dots, m\}$, $p \in \{1, \dots, N\}$. For instance, the encoding of the configuration of $c_0 = \langle\langle 1, w_{red}\rangle, \langle 2, w_{red}\rangle, \langle 3, w_{blue}\rangle\rangle$ is the formula

$$\underbrace{\neg x_{1.1.0} \wedge \neg x_{2.1.0}}_{red} \wedge \underbrace{\neg x_{1.2.0} \wedge \neg x_{2.2.0}}_{red} \wedge \underbrace{\neg x_{1.3.0} \wedge x_{2.3.0}}_{blue}$$

We are ready now to encode a transition rule.

Symbolic Representation of the Transition Rules A CA is usually given in form of a table: each row is transition rule for one possible global state of the neighborhood of a generic cell. Let us call $\mathcal{R}_p$ the set of rows of the table relative to a cell in position p. Let us assume that the neighborhood is $v_1, \dots, v_n$. Let $R \in \mathcal{R}_p$ be a rule that, at time t, operates on the neighborhood of a cell p, namely $\langle\langle p, w\rangle, \langle (p + v_1, w_1\rangle, \dots, \langle p + v_n, w_n\rangle\rangle$, and that updates its state into w'. Then, the encoding of R is the formula

$$Cod_r(R, p, t) \doteq Cod_w(w, p.t) \wedge \bigwedge_{i=1}^{n} Cod_w(w_i, (p + v_i).t) \wedge Cod_w(w', p.(t + 1))$$

We can extend this encoding to $\mathcal{R}_p$ in the natural way:

$$Cod_R(\mathcal{R}_p, t) \doteq \bigvee_{R \in \mathcal{R}_p} Cod_r(R, p, t)$$

Using this disjunctive formula we can express one evolution step without having to specify the initial configuration (we let open all possible choices of rules in $\mathcal{R}_p$). Note that to obtain a total transition function with respect to the set of variables used in the encoding we need to add identity rules

Symbolic Representation of the CA-Evolution Specifically, the formula $Ev^{\mathcal{A}}(N,k)$ that describes all possible evolutions in k steps of a CA with N cells is defined as follows

$$Ev^{\mathcal{A}}(N,k) \doteq \bigwedge_{t=0}^{k-1} \bigwedge_{i=1}^{N} Cod_R(\mathcal{R}_i, t)$$

Note that the formula $Ev^{\mathcal{A}}(N,k)$ is not in conjunctive normal form (CNF). The following properties formalize the connection between the evolution of a CA and the formula $Ev^{\mathcal{A}}(N,k)$.

Proposition 1. *Given a CA $\mathcal{A}$, every assignment ρ satisfying of the formula $Ev^{\mathcal{A}}(N,k)$ represents a possible evolution $\{c_t\}_{t\geq 0}$ of $\mathcal{A}$ such that ρ satisfies $Cod(c_t,t)$ for any $t \geq 0$.*

As a consequence, we have the following link between k-reachability and satisfiability of $Ev^{\mathcal{A}}(N,k)$.

Theorem 1. *Given a CA $\mathcal{A}$ and two configurations c and c', c' is reachable in k-steps from c if and only if the formula*

$$REACH_k \doteq Cod_c(c,0) \wedge Ev^{\mathcal{A}}(N,k) \wedge Cod_c(c',k)$$

is satisfiable.

As a final remark, it is easy to check that the size of the encoding is polynomial in the size of the cellular space, size of the neighborhood and in the number of steps taken into consideration.

Example 2. Let $\mathcal{A}$ be a 1-CA, with $S = \{0,1\}$, circular boundary conditions and neighborhood $I = \{-1\}$ (i.e. we only look at the left neighbor's cell) and rule described in Fig. 2. In the left table of Fig. 2 we use the variable x^{it} to denote the value of cell i a time t. The column x^{it+1} denotes the new state of cell i at time $t+1$. The right table of Fig. 2 shows the corresponding encoding of the local rule when interpreting 0 and 1 as truth values. A configuration $c = \langle 0,1 \rangle$ with 2 cells at time 0 can be encoded as the conjunction of literals $\neg x_{1.0} \wedge x_{2.0}$. Thus, the evolution from $t=0$ to $t=1$ of cell c can be represented as follows

$$\neg x_{1.0} \wedge x_{2.0} \wedge \left(\begin{array}{c} \neg x_{1.0} \wedge \neg x_{2.0} \wedge x_{2.1} \\ \vee \\ \neg x_{1.0} \wedge x_{2.0} \wedge \neg x_{2.1} \\ \vee \\ x_{1.0} \wedge \neg x_{2.0} \wedge \neg x_{2.1} \\ \vee \\ x_{1.0} \wedge x_{2.0} \wedge x_{2.1} \end{array} \right) \wedge \left(\begin{array}{c} \neg x_{2.0} \wedge \neg x_{1.0} \wedge x_{1.1} \\ \vee \\ \neg x_{2.0} \wedge x_{1.0} \wedge \neg x_{1.1} \\ \vee \\ x_{2.0} \wedge \neg x_{1.0} \wedge \neg x_{1.1} \\ \vee \\ x_{2.0} \wedge x_{1.0} \wedge x_{1.1} \end{array} \right) \wedge \neg x_{1.1} \wedge \neg x_{2.1}$$

x^{i-1t}	x^{it}	x^{it+1}
0	0	1
0	1	0
1	0	0
1	1	1

$$\Rightarrow \quad \begin{array}{c} (\neg x_{i-1.t} \wedge \neg x_{i.t} \wedge x_{i.t+1}) \\ \vee \\ (\neg x_{i-1.t} \wedge x_{i.t} \wedge \neg x_{i.t+1}) \\ \vee \\ (x_{i-1.t} \wedge \neg x_{i.t} \wedge \neg x_{i.t+1}) \\ \vee \\ (x_{i-1.t} \wedge x_{i.t} \wedge x_{i.t+1}) \end{array}$$

Fig. 2. Encoding of the local rule.

As expected, the valuation

$$v : \{x_{1.0} \mapsto F, x_{2.0} \mapsto T, x_{1.1} \mapsto F, x_{2.1} \mapsto F\}$$

satisfies the resulting propositional formula. If we do not specify the final configuration, a solution to the SAT-problem

$$\neg x_{1.0} \wedge x_{2.0} \wedge \left(\begin{array}{c} \neg x_{1.0} \wedge \neg x_{2.0} \wedge x_{2.1} \\ \vee \\ \neg x_{1.0} \wedge x_{2.0} \wedge \neg x_{2.1} \\ \vee \\ x_{1.0} \wedge \neg x_{2.0} \wedge \neg x_{2.1} \\ \vee \\ x_{1.0} \wedge x_{2.0} \wedge x_{2.1} \end{array} \right) \wedge \left(\begin{array}{c} \neg x_{2.0} \wedge \neg x_{10} \wedge x_{1.1} \\ \vee \\ \neg x_{2.0} \wedge x_{1.0} \wedge \neg x_{1.1} \\ \vee \\ x_{2.0} \wedge \neg x_{1.0} \wedge \neg x_{1.1} \\ \vee \\ x_{2.0} \wedge x_{1.0} \wedge x_{1.1} \end{array} \right)$$

allows us to compute the successor configuration by simply projecting the resulting valuation on variables stamped with time index 1. In the following section we will formalize more precisely how to specify reachability properties using SAT-encoding of CA-evolutions.

4 SAT-based Qualitative Reasoning

The formula $Ev^{\mathcal{A}}(N, k)$ represents an encoding of all possible CA-evolutions of length k independently from any initial or target configuration. This property allows us to encode in a modular way several interesting properties of CAs in terms of satisfiability of a propositional formula. In the rest of the section we will discuss some examples.

Reachability Given an initial configuration c and a configuration c', we can decide whether c' is reachable in k-steps from c by solving the satisfiability problem for the formula

$$REACH_k \doteq Cod_c(c, 0) \wedge Ev^{\mathcal{A}}(N, k) \wedge Cod_c(c', k)$$

Actually, we can also compute all configurations reachable in at most k-steps. We first solve the satisfiability problem for the formula

$$CREACH_k \doteq Cod_c(c, 0) \wedge Ev^{\mathcal{A}}(N, k)$$

and then extract the configurations c_t for $0 \leq t \leq k$ from the resulting satisfying assignment ρ.

Note that the formula $CREACH_k$ is always satisfiable if the rule table contains a totally defined rule, otherwise we have to choose accurately the initial state.
$(C)REACH_k$ can be refined in order to be satisfiable only if the evolution is acyclic. The acyclicity test $ACYCLIC_k$ amounts to require that the assignment to predicates at time t is distinct from all assignments at time $t' < t$ for any pair of values of t and t' between 0 and k. Thus, for finite CA we can explore all the reachable configurations by *iterative deepening* on k until the $ACYCLIC_k$ fails. This way we can solve reachability problems and compute the reachability set of a CA.

Inverse Reachability Compared to approaches based on simulation, a distinguishing feature of our encoding is that the formula $Ev^{\mathcal{A}}(N,k)$ is independent from the search strategy (e.g. forward exploration/backward exploration). we adopt for the analysis of a CA. This feature makes easy the encoding of *inverse* reachability problems in terms of reachability. For instance, given a (sub)configuration c and a configuration c', we can decide whether c' is a predecessor of c by solving the satisfiability problem for the formula

$$PEP \doteq Cod_c(c',0) \wedge Ev^{\mathcal{A}}(N,1) \wedge Cod_c(c,1)$$

Note that, if c is a subconfiguration, in its encoding all unspecified cells will be represented with predicates without constraints on its truth values. Similarly, given a (sub)configuration c and a configuration c', we can decide whether c' is a predecessor in k-steps of c by solving the satisfiability problem for the formula

$$PREP_k \doteq Cod_c(c',0) \wedge Ev^{\mathcal{A}}(N,k) \wedge Cod_c(c,k)$$

Finally, as for forward reachability, we can also *compute* a possible trace in the CA-evolution (and the corresponding initial state) of k steps that leads to an encoded configuration c at time k. We first solve the satisfiability problem for the formula

$$IREACH_k \doteq Ev^{\mathcal{A}}(N,k) \wedge Cod_c(c,k)$$

and then extract the configurations c_t for $0 \leq t \leq k$ from the resulting satisfying assignment ρ. In order to find the set of all predecessors of a configuration c we can use the following procedure: (1) set F to $IREACH_1$; (2) solve the satisfiability problem for the formula F (that gives us as a result *one* possible predecessor); (3) if the problem is unsatisfiable exit the procedure, otherwise (4) extract the formula G corresponding to the computed predecessor, set F to $F \wedge \neg G$ and go back to (2).

4.1 Goal-driven SAT-encoding

Inverse reachability is the more difficult problem among the one listed in the previous section. This is due to the non-determinism in the computation of the

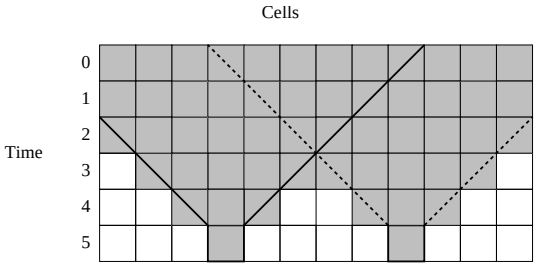

Fig. 3. Example of cone of influence associated to a given final subconfiguration.

$$cone(i,0) = \{\langle i,0\rangle\}$$
$$cone(i,t) = \{\langle i,t\rangle\} \cup cone(i,t-1) \cup \bigcup_{j=1}^{n} cone(i+v_j,t-1), \text{ for } t>0$$

Fig. 4. Definition of the cone of influence of a given cell.

preimage of a given configuration. To reduce the complexity of the SAT-solving procedure we can try to reduce the size of the SAT-formula encoding the CA-evolution and specialize it to the goal we are looking for. For example, suppose that for a 1-CA with N cells we want to compute the initial (sub)configurations that lead to a certain state for cell i in k steps. To optimize the encoding of this problem, we can apply a version of the *cone of influence* introduced in [1] to statically compute the set of variables that influence the evolution of cell i. In Fig. 3 an example of *cone of influence* is presented for 2 cells after 5 evolution-step of a 1-CA with neighborhood of unitary radius. The gray zone shows the union of the useful cones for the considered cells. The Fig. 4 we define a procedure *cone* for computing the cone of influence associated to a given cell i at time t. The procedure only depends on the definition of the neighborhood. Our goal is to reduce the number of variables in the encoding of the CA-evolution by choosing only necessary cells for the encoded properties. Suppose that c is the subconfiguration for which we want to compute the predecessors in k-steps. Then, the set of variables computed by the algorithm in Fig. 4 can be used during the construction of the SAT-formula as follows. We first construct the cone of influence for all cells i contained in c. Then, we generate the formula $\mathbf{Cod_r}(\mathcal{R}, i', t')$ if and only if $\langle i', t'+1\rangle \in cone(i,t)$. The quality of this heuristic clearly depends on the *locality* of the neighborhood and on the final subconfiguration.

To limit the number of variables in the SAT-formula, we can exploit the fact that *boundary* cells (i.e. cells that encode the boundary of the cellular space) never change state. Thus, we only need to encode boundary cells at time zero and refer to this encoding in every step of the construction of the SAT-formula. This optimization preserves the correctness of the encoding.

In the following section we will discuss a practical evaluation of the proposed SAT-based methodology and related heuristics/optimizations.

5 Experimental Results

In order to test the effectiveness of the SAT-based analysis we have performed several experiments using the SAT-solver zChaff [14]. zChaff implements the Davis-Putnam algorithm [5] and it is considered as one of the fastest existing solvers. In general zChaff manages formulas with about 10^6 propositional variables and about 10^7 clauses. In some preliminary experiments presented in [4] we have compared zChaff with other solvers like ICS [8], SIMO [16] and HeerHugo [7] on problems related to CAs. zChaff always gave us the best results in terms of execution time. The input for zChaff is a CNF-formula written in DIMACS format. By using the structure-preserving algorithm of [15], we have built a front end to put the formula resulting from the encoding of a CA-evolution in CNF. The algorithm makes use of a polynomial number of auxiliary variables (one per each row of a CA-table). All experiments are performed on a Pentium4 2 GHz, with 1Gb of RAM.

5.1 Tested Example

As main example we have considered a solution to the Firing Squad Synchronization Problem (FSSP). FSSP was introduced by Moore in [13]. One considers here a finite ordered line of n finite-state machines. At time 0, the leftmost cell is distinguished (general) from the others (soldiers). These machines work synchronously; the state of a machine i at time $t+1$ depends only on the states at time t of the machines $i-1$, i and $i+1$. The problem is to define finite sets of states and transition rules so that all machines enter for the first time a distinguished state (fire) at the very same moment. This problem can be solved by defining a 1-CA with n cells representing the firing squad and the general. Mazoyer [12] has given a six-state (plus a cell for the boundary of the cellular space) minimal time solution in which the general creates two *waves* that propagates through the squad at different speed so as to reach a solution in exactly $2 * \#cells - 2$. This problem is thus adequate for testing *bounded reachability* problems. Mazoyer's CA is defined via 120 interesting rows (the total transition relation has 7^3 rows, the remaining 223 rows do not change the cell state).

5.2 Tested Properties

In Table 1 we illustrate the type of reachability properties we have tested on FSSP. Specifically, we have considered reachability problems in which either the initial and final configuration are completely specified or part of them are left unconstrained. For instance, $I^{-n}F$ denotes a reachability problem in which n cells of the initial configuration are left unconstrained (i.e. we considered a subconfiguration of the initial configuration); F denotes a problem in which only the final state is specified. The properties $nF + B$ and $F + B + nL$ listed in Table 1 are related to special tricks we used to exploit an heuristic called VSIDS of zChaff. The heuristic VSIDS is used to choose a starting variable for the resolution algorithm between those that appear most frequently in the

Added to $Ev^{\mathcal{A}}$	**Description**
I	Initial configuration (i.e. *CREACH*).
I^{-n}	Initial subconfiguration in which n cells are unconstrained.
F	Final configuration (i.e. *IREACH*)
F^1	One cell of the final configuration.
B	Boundary cells.
$I + F$	Initial configuration (i.e. *REACH*).
$F + B$	Final configuration and boundary cells.
$I^{-n} + F$	Initial configuration without n cells and a final configuration.
$F^1 + B$	Only one cell of the final configuration and boundary cells
$nF + B$	Final configuration with n-copies of formula F.
$F + B + nL$	$F + B$ and n-copies of the last step of the evolution formula.

Table 1. List of reachability properties considered in the experiments.

formula. In order to exploit this heuristic we can either put n-copies of the formula encoding F (property $nF + B$) or put n-copies of the last step of the formula representing the evolution (i.e. the clauses leading to variables occurring in F) (property $F + B + nL$). This way when computing predecessors of a configuration we force the SAT-solver to choose the variables that encode the final configuration, i.e., those with the smaller number of occurrences in $Ev^{\mathcal{A}}$ but with trivial truth assignment.

5.3 Outcomes of the Experiments

All the experiments require a preliminary compilation phase in which the SAT-formula is built up starting from a CA rule table. The time required for the biggest example is around 20 minutes due to the huge size of the resulting output. In the following we will focus however on the performance of the solver on SAT-formula of different size and on properties taken from Table 1.

***I + F* and *I^{-n} + F* Properties** In a first series of experiments we have tested $I+F$-like properties on the CA-solution to FSSP. The results are shown in Table 2. The final configuration considered here is the one in which all soldiers in the firing squad have received the fire command. For this kind of problems, the size of the formulas that zChaff manages to solve scales up smoothly to formulas with one million variables. For instance, on a cellular space of dimension 70 and with 138 evolution steps it requires 1 minute to check that the CA solves FSSP. We considered then problems of the form $I^{-n}F$. Since there might be several initial states (legal or illegal) containing the subconfiguration I^{-n} and leading to the same final state F, the resulting SAT problem becomes more difficult. As expected, on this new kind of problems the performance of zChaff decreases with the number of cells n removed from I. As an example, for a CA with 15 cells (hence 28 steps) it takes more $11m$ to solve the $I^{-4} + F$ problem.

Problem	#Cells	#Steps	Input(MB)	#Vars	#Clauses	ExTime
$I + F$	15	28	12.18	51708	655713	3s
$I + F$	29	56	51.76	199842	2535241	13s
$I + F$	40	78	101.8	383883	4870563	25s
$I + F$	50	98	165.57	602853	7649203	39s
$I + F$	70	138	340.21	1118393	15079683	1m 22s
$I^{-1} + F$	15	28	12.18	51708	655710	3s
$I^{-2} + F$	15	28	12.18	51708	655707	69s
$I^{-3} + F$	15	28	12.18	51708	655704	65s
$I^{-4} + F$	15	28	12.18	51708	655701	30m53s

Table 2. Experiments on $I + F$-like problems.

Problem	#Cells	#Steps	Input(MB)	#Vars	#Clauses	ExTime
I	15	28	12.18	51708	655668	3s
I	29	56	51.76	199842	2535154	13s
I	50	98	165.57	602853	7649053	40s
I	70	138	340.21	1118393	15079473	1m 06s
I^{-1}	15	28	12.18	51708	655665	3s
I^{-2}	15	28	12.18	51708	655662	7m 21s
I^{-3}	15	28	12.18	51708	655659	4s
I^{-4}	15	28	12.18	51708	655656	10s
I^{-5}	15	28	12.18	51708	655653	3m 22s
I^{-7}	15	28	12.18	51708	655647	35m 36s

Table 3. Experiments on I and I^{-n} problems.

***I* and I^{-n} Properties** In a second series of experiments we have tested I-like properties that can be used to *compute* reachable states. The results are shown in Table 3. Unexpectedly, the behavior of zChaff is quite irregular with respect to the growth of the size of the SAT formulas. The average of the execution times tends to grow exponentially with the number of cells removed from the initial configuration until the problem becomes trivial (i.e. when we do not have neither I nor F). Other examples we tested in [4] did not suffer from this anomaly.

***F* Properties** In the third series of experiments we have considered different types of F-like properties that can be used to compute *predecessor configurations* of a given final configuration. This is a hard problem in general. As expected, we had to reduce the size of the SAT-formulas in order to get reasonable execution times. In Table 4 we have considered inverse reachability starting from F. For a cellular space of dimension 10 it takes about $9m$ to solve the problem F. Adding a constraint on the boundary cells, i.e. property $F + B$, we dramatically decrease the execution time ($2m$). In this example zChaff infers the correct initial

Problem	#Cells	#Steps	Input(MB)	#Vars	#Clauses	ExTime
F	10	18	5	22173	281010	9m 40s
$F + B$	10	18	5	22173	281013	2m 27s
$F + B + 2L$	10	18	5	22173	296623	35s
$10F + B$	10	18	5	22173	281043	39s
$400F + B$	10	18	5	22173	292983	1m 50s

Table 4. Experiments on F-like problems.

Problem	C.o.I.	#Cells	#Steps	Input(MB)	#Vars	#Clauses	ExTime
$F^1 + B$		10	18	5	22173	280987	1m 48s
$F^1 + B$	$\surd$	10	18	4	16773	241961	1m 59s
$F^1 + B + 10L$		10	18	5	22173	296605	2m 05s
$F^1 + B + 10L$	$\surd$	10	18	4	16773	257561	1m 40s
$F^1 + B + 30L$		10	18	5	22173	330962	54s
$F^1 + B + 30L$	$\surd$	10	18	4	16773	288791	59s
$100F^1 + B$		10	18	4	22173	281283	2m20s
$100F^1 + B$	$\surd$	10	18	4	16773	241961	2m

C.o.I.=Reduction of the SAT-formula via the Cone of Influence

Table 5. Experiments on F^1-like problems.

configuration. The execution time further improves when forcing zChaff to select the variables occurring in F with the trick discussed at the beginning of this section. Specifically, when duplicating the last step of the formula $Ev^{\mathcal{A}}$ zChaff takes $35s$ to solve the same problem (see prop. $F + B + 2L$ in Table 4). Similar results are obtained by simply copying F 10 times in the input formula given to zChaff (see prop. $10F + B$ in Table 4). Tuning the heuristics might be difficult (without modifying the code of zChaff) as shown by further experiments (like $100F + B$) where the performance gets worse again.

***F^1* Properties** In order to study the effectiveness of the cone of influence reduction we have also considered $F^{-1} + B$ properties as shown in Table 5. The use of this reduction alone does not improve the execution time much (see $F^1 + B$ with and without C.o.I. in Table 5). However, when coupled with the n-copy of the last step of $Ev^{\mathcal{A}}$ (to force the selection of variables in F) then it seems to work (see $F^1 + B + 10L$ with and without C.o.I. in Table 5). Copying the formula F does not seem to work well in this examples. Again tuning the parameters used in heuristic like the number of copies n in $\ldots + nL$ might be difficult by simply using zChaff as a black box.

Problem	#Cells	#Steps	#Vars	#Clauses	Input(MB)	ExTime
I	10	18	738044	12534004	237.00	5s
I	15	1	61564	1044495	17.26	5s
I	15	2	123064	2088990	36.02	9s
I	15	3	123064	3133485	56.23	14s
I	15	10	651064	10445014	197.70	48s
I	15	15	922564	15667489	299.00	1m54s
I	15	17	1045564	17756479	341.00	2m00s
I	15	20	1230064	20889964	410.41	ABORT

Table 6. Experiments on a CA with 4096 table rows.

5.4 Other Examples

In the last series of experiments we have randomly generated a CA with more than 4000 table rows and tested on it I-properties of increasing size. The aim here was to reach the limit of zChaff w.r.t. number of variables and clauses generated by the encoding on an easy problem. As shown in Table 6, zChaff gets in trouble when the formula has more than one million variables and about 20 millions of clauses (15 cells, 20 steps), while it can handle problems with 17 millions clauses (15 cells, 17 steps). This kind of analysis can be useful to evaluate the size of CAs we can handle with non-specialized SAT solvers.

6 Conclusions and Related Work

Although several CAs programming and simulation tools have been developed (see e.g. the survey of [22]), we are not aware of general frameworks for performing qualitative analysis of CAs automatically. In this paper we have proposed a SAT-based methodology for attacking this problem. One of the advantages of the proposed method is that, once the encoding of the CA-evolution has been computed, several different reachability problems can be formulated as simple propositional queries to a SAT-solver. The formula encoding the evolution can then be reused in a modular way (we can shrink or extend it easily and attach to it different initial/final configurations). Hard problems like inverse reachability can be attacked then by using modern SAT-solvers like zChaff that seems to perform well on problems with millions of variables and clauses.

Although this seems a new approach for checking properties of CAs, SAT technology is widely used for computer aided verification of hardware and software design. As an example, tools for Bounded Model Checking like nuSMV [1] automatically generate an *unfolding* of the transition relation of a high level description of a reactive system and then use a SAT-solver to test LTL properties. The reason why we did not resort to existing SAT-based verification tools like nuSMV is that we wanted to have complete control over the SAT-formula generated by the encoding of a CA-evolution. This way we were able to test different

kind of properties and heuristics on zChaff directly.

In our preliminary experiments we have obtained interesting results for CAs of reasonable size (e.g. 70 cells, 140 steps, 120 rules). We believe that it might be possible to manage larger problems by a specialization of the SAT-solving algorithm (and, especially, of its heuristics) that could benefit from structural properties of CAs. This might be an interesting future direction for our research.

References

1. A. Biere, E.M. Clarke, R. Raimi, Y. Zhu. *Verifiying Safety Properties of a Power PC Microprocessor Using Symbolic Model Checking without BDDs*, CAV '99, Lecture Notes in Computer Science, 60-71, 1999.
2. E. Burks. *Essays on Cellular Automata*, University of Illinois Press, 1972.
3. E.F.Codd. *Cellular Automata*, Academic Press, New York, 1968.
4. M. D'Antonio. *Analisi SAT-based di Automi Cellulari*, Tesi di laurea, Dip. di Informatica e Scienze dell'Informazione, Università di Genova, Marzo 2004.
5. M. Davis, H. Putnam. *A computing procedure for quantification theory*, Journal of the Association for Computing Machinery, 1960.
6. F. Green. *NP-Complete Problems in Cellular Automata*, Complex Systems, 1:453-474, 1987.
7. J. Groote, J. Warners *The propositional formula checker HeerHugo*, Journal of Automated Reasoning, 24(1-2):101-125, 2000.
8. The ICS home page: http://www.icansolve.com
9. J. Kari. *Cryptosystems based on reversible cellular automata*, Preprint, 1992.
10. J. Kari. *Reversability of 2D cellular automata is undecidable*, Physica, Vol. D 45:379-385, 1990.
11. C. Langton. *Studying artificial life with cellular automata*, Physica D, 22:120-140, 1986.
12. J. Mazoyer. *A six-state minimal time solution to the firing squad synchronization problem*, Theoretical Computer Science, 50(2):183-240, Elsevier, 1987.
13. E.F. Moore. *Sequential machines* in: E.F. Moore *Selected Papers*, Addison-Wesley, Reading, 1964.
14. M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik. *Chaff: Engineering an Efficient SAT Solver*, Proceedings of the 38th Design Automation Conference (DAC'01), 2001.
15. D. A. Plaisted and S. Greenbaum. A Structure Preserving Clause Form Translation. Journal of Symbolic Computation, 2(3):293–304, 1986.
16. The SIMO web page: http://www.mrg.dist.unige.it/ sim/simo/
17. K. Sutner. *On the computational complexity of finite cellular automata*, JCSS, 50(1):87-97, 1995.
18. J. von Neumann. *Theory of Self-Reproducing Automata*, University of Illinois Press, edito e completato da A.W. Burks, 1966.
19. S. Wolfram. *Universality and complexity in cellular automata*, Physica D, 10:1-35, 1984.
20. S. Wolfram.*Computation Theory of Cellular Automata*, Communications in Mathematical Physics, 96:15-57, November 1984.
21. S. Wolfram. *Theory and Applications of Cellular Automata*, World Scientific, 1986.
22. T. Worsch. *Programming Environments for Cellular Automata*,Technical report 37/96, Universitat Karlsruhe, Fakultat fur Informatik, 1996.