SAT-Based Analysis of Cellular Automata

Massimo D'Antonio and Giorgio Delzanno

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

Abstract. In the last decade there has been a dramatic speed-up for algorithms solving the Satisfiability (SAT) problem for propositional formulas, a well known NP-complete problem. In this paper we will investigate the applicability of modern SAT *solvers* to the qualitative analysis of Cellular Automata. For this purpose we have defined an encoding of the *evolution* of Cellular Automata into formulas in propositional logic. The resulting formula can be used to test *forward* and *inverse reachability* problems for the original Cellular Automata in a modular way. In the paper we will report on experimental results obtained by applying the SAT-solver zChaff to reachability problems for classical examples of Cellular Automata like the Firing Squad Synchronization Problem.

1 Introduction

Several tools have been used to animate specifications of Cellular Automata (CAs) e.g., for traffic and artificial life processes simulation (for a survey, see e.g., [15]). Simulation amounts to compute all configurations reachable from an initial one, a simple problem for a *deterministic* CA. Differently from plain simulation, a *qualitative analysis* of the behavior of a CA can be a hard problem to solve. Problems like reachability of a subconfiguration or existence of a predecessor configuration have been shown to be exponentially hard in the worst case [4,13,14]. In recent years practical solutions to large instances of known NP-complete problems like Satisfiability (SAT) of propositional formulas have been made possible by the application of specialized search algorithms and pruning heuristics. Since 1991 the problems solvable by SAT solvers have grown from 100 to 10,000 variables. On the basis of these results, researchers from various communities have started encoding real-world problems into SAT. This has lead to breakthroughs in planning and model-checking of hardware circuits [1]. The connection between the complexity of some interesting problems for CAs and SAT (see e.g. [4]) suggests us a possible new application of all these technologies.

In this paper we will investigate the applicability of SAT solvers to the *qualitative analysis* of CAs. Specifically, following the *Bounded Model Checking* (BMC) paradigm 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 forward and backward (inverse) reachability

P.M.A. Sloot, B. Chopard, and A.G. Hoekstra (Eds.): ACRI 2004, LNCS 3305, pp. 745–754, 2004.

as a satisfiability problem. As a conceptual contribution, we obtain an effective method for the analysis of CAs in which we can plug in SAT modern solvers like [5,6,11,9]. As a preliminary experiment, we have studied several reachability problems for Mazoyer's solution to the Firing Squad Synchronization Problem [7]. 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 zChaff [9]. zChaff 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 will give some preliminary notions on Cellular Automata and the SAT problem. In Sections 3 and 4 we will define an encoding of the evolution of a CA and of its qualitative problems into a SAT formula. In Section 5 we will discuss experimental results obtained by using the SAT solver zChaff. In Section 6 we will discuss related works and future directions of our research.

2 Preliminaries

Cellular Automata. 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 $\mathcal{A}$, with cardinality n; $\delta : \ S^{n+1} \rightarrow S$ is the transition rule of $\mathcal{A}$. The definition of δ takes into account the cell and its neighborhood. A *configuration* 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.

The Satisfiability Problem. The Satisfiability problem (SAT) is a decision problem considered in complexity theory. An instance of the problem is defined

by a Boolean expression written using only the connectives $\wedge$ (*and*), $\vee$ (*or*), $\neg$ (*not*), and propositional variables. The question is: given the expression, is there some assignment of *true* and *false* values to the variables that will make the entire expression true? In mathematics, a formula of propositional logic is said to be satisfiable if truth-values can be assigned to its free variables in a way that makes the formula true. The class of satisfiable propositional formulae is NP-complete, as is that of its variant 3-Satisfiability (3SAT). This means that the best known algorithm that solves this problem has worst case exponential time complexity unless the complexity class P is equal to NP. Other variants, such as 2-Satisfiability and Horn-satisfiability, can be solved by efficient algorithms.

Qualitative Analysis of CAs. 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. This because configurations might have a *branching past*. For instance, the Predecessor Existence Problem

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

is NP-complete for d-CAs with $d > 1$ (see [12]). The proof of NP-completeness can be given via a reduction from 3SAT. In this paper we are interested however in the inverse reduction (i.e. from CA to SAT), our goal being the definition of an effective method for the analysis of CAs.

3 From Cellular Automata to Propositional Logic

In this section we will describe how to represent a finite number of evolution steps of a CA in propositional logic. The resulting formula will be used later to specify several different qualitative analyses of a CA.

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). Every state represented in binary can be naturally encoded as a propositional formula as follows. Let ℓ be a label and w a sequence of m bits $b_1 \ldots b_m$. Furthermore, let . be an operator for concatenating labels. Then, we define the formula encoding as follows:

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

Let us assume that the cellular space is linearized into the range $1, \ldots, N$. Then, a configuration is simply a tuple $c = \langle \langle 1, w'_1 \rangle, \ldots, \langle N, w'_N \rangle \rangle$, where $w'_i \in S'$. The encoding of c at instant t is defined then as follows:

$$Cod_c(c, t) \doteq Cod_w(w'_1, 1.t) \wedge \ldots \wedge Cod_w(w'_N, N.t)$$

Thus, $Cod_c(c,t)$ gives rise to a conjunction of *literals* over the set of predicate symbols $x_{b.p.t}$ where $b \in \{1,\ldots,m\}$, $p \in \{1,\ldots,N\}$.

Transition Rules. A CA local rule is usually given in form of a table: each row is a transition rule for a generic cell in function of its state and one possible global state of the neighborhood. 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 $v_1,\ldots,v_n$. Let $R \in \mathcal{R}_p$ be a rule that, at time t, operates on the neighborhood of a cell p, namely $\langle\langle p,w\rangle,\langle p+v_1,w_1\rangle,\ldots,\langle p+v_n,w_n\rangle\rangle$, 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 $\mathcal{R}_p$).

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

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

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 that satisfies the propositional 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*

$$REACH_k \;\doteq\; Cod_c(c,0) \wedge Ev^{\mathcal{A}}(N,k) \wedge Cod_c(c',k) \qquad (1)$$

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.

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. By solving of the formula (1) we can decide whether c' is reachable in k-steps from c. To 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.

$(C)REACH_k$ can be refined in order to be satisfiable only if the evolution is acyclic, i.e. 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 acyclicity test fails.

Inverse Reachability With our encoding it's easy to encode the *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 in k-steps of c by solving the satisfiability problem for the formula

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

So we can find a solution for the Predecessor Existence Problem (PEP) solving the formula $PREP_1$. 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

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 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 a cell, if we want to study its final or initial state. To limit the number of variables in the SAT-formula, we can also exploit the fact that *boundary* cells 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 [9]. The input for zChaff is a CNF-formula written in DIMACS format. By using the structure-preserving algorithm of [10], 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). Mazoyer [7] has given a six-state (plus a cell for the boundary of the cellular space) minimal time solution defined via 120 interesting rows.

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 encoded 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 encoded. 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 [9]. The heuristic VSIDS is used to choose a starting variable for the resolution algorithm between those that appear most frequently in the formula. In order to force the SAT-solver to choose the variables that encode a certain configuration, we can either put n-copies of this encoding (like properties $nF + B$) or put n-copies of the step of the formula representing the evolution that contains these variables (like property $F + B + nL$). This way when computing predecessors of

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

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.
$I + F$	Initial and final 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.

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 Experimental Evaluation

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 [Table 2]. For this kind of problems, the size of the formulas that zChaff manages to solve scales up smoothly to formulas with one million variables. 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.

I and I^{-n} Properties [Table 3]. In a second series of experiments we have tested I-like properties that can be used to *compute* reachable states. 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 [2] did not suffer from this anomaly.

F Properties [Table 4]. In the third series of experiments we have considered different types of F-like properties that can be used to compute *predecessor configurations*, one configuration by step, of a given final configuration. This is a hard problem in general. As expected, we had to reduce the size of the

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

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

SAT-formulas in order to get reasonable execution times. 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 ($2m$). 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 Similar results are obtained by simply copying F 10 times in the input formula given to zChaff. Tuning the heuristics might be difficult (without modifying the code of zChaff) as shown by further experiments (like $400F + B$) where the performance gets worse again.

F^1 **Properties [Table 5].** In order to study the effectiveness of the cone of influence reduction we have also considered $F^1 + B$ properties. The use of this reduction alone does not improve much the execution time. 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

Table 4. Experiments on F-like problems.

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 5. Experiments on F^1-like problems.

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

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

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.

5.4 Other Examples

In another series of experiments we have randomly generated a 1-CA with 4096 table rows (16 states and nearest-neighbor) 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. 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) and 1 million of variables. 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 [15]), 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. 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 [1]. In our preliminary experiments we have obtained interesting results for CAs of reasonable size (e.g. 70 cells, 140 steps, 120 rule rows). 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. M. D'Antonio. *Analisi SAT-based di Automi Cellulari*, Tesi di laurea, Dip. di Informatica e Scienze dell'Informazione, Università di Genova, Marzo 2004.
3. M. Davis, H. Putnam. *A computing procedure for quantification theory*, Journal of the Association for Computing Machinery, 7:201-215, 1960.
4. F. Green. *NP-Complete Problems in Cellular Automata*, Complex Systems, 1:453-474, 1987.
5. J. Groote, J. Warners *The propositional formula checker HeerHugo*, Journal of Automated Reasoning, 24(1-2):101-125, 2000.
6. The ICS home page: http://www.icansolve.com
7. J. Mazoyer. *A six-state minimal time solution to the firing squad synchronization problem*, Theoretical Computer Science, 50(2):183-240, Elsevier, 1987.
8. E.F. Moore. *Sequential machines* in: E.F. Moore *Selected Papers*, Addison-Wesley, Reading, 1964.
9. 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.
10. D. A. Plaisted and S. Greenbaum. *A Structure Preserving Clause Form Translation* Journal of Symbolic Computation, 2(3):293-304, 1986.
11. The SIMO web page: http://www.mrg.dist.unige.it/ sim/simo/
12. K. Sutner. *On the computational complexity of finite cellular automata*, JCSS, 50(1):87-97, 1995.
13. S. Wolfram. *Universality and complexity in cellular automata*, Physica D, 10:1-35, 1984.
14. S. Wolfram.*Computation Theory of Cellular Automata*, Communications in Mathematical Physics, 96:15-57, November 1984.
15. T. Worsch. *Programming Environments for Cellular Automata*,Technical report 37/96, Universitat Karlsruhe, Fakultat fur Informatik, 1996.