Formal Methods for the Analysis and Synthesis of Nanometer-Scale Cellular Arrays

JOSEP CARMONA and JORDI CORTADELLA
Universitat Politècnica de Catalunya
YOUSUKE TAKADA
Canopus Company Ltd.
and
FERDINAND PEPER
National Institute of Information and Communications Technology

Nanometer-scale structures suitable for computing have been investigated by several research groups in recent years. A common feature of these structures is their dynamic evolution through cascaded local interactions embedded on a discrete grid. Finding configurations capable of conducting computations is a task that often requires tedious experiments in laboratories. Formal methods—though used extensively for the specification and verification of software and hardware computing systems—are virtually unexplored with respect to computational structures at atomic scales. This paper presents a systematic approach toward the application of formal methods in this context, using techniques like abstraction, model-checking, and symbolic representations of states to explore and discover computational structures. The proposed techniques are applied to a system of CO molecules on a grid of Copper atoms, resulting in the design of a complete library of combinational logic gates based on this molecular system. The techniques are also applied on (more general) systems of cellular automata that employ an asynchronous mode of timing. The use of formal methods may narrow the gap between Physical Chemistry and Computer Science, allowing the description of interactions of nanometer scale systems on a level of abstraction suitable to devise computing devices.

This article is an extended version of the paper "From Molecular Interactions to Gates: A Systematic Approach" in Proceedings of the International Conference on Computer-Aided Design (ICCAD'06) 891–898.
This work has been supported by the project FORMALISM (TIN2007-66523).
Authors' addresses: J. Carmona, J. Cortadella, Software Department, Universitat Politècnica de Catalunya, Jordi Girona, 1-3,08034, Barcelona, Spain; Y. Takada, Canopus Co., Ltd., 1-2-4 Murotani, Nishi-ku, Kobe 651-2241, Japan; F. Peper, National Institute of Information and Communications Technology, Nano ICT Group, 588-2 Iwaoka, Iwaoka-cho, Nishi-ku, Kobe 651-2492, Japan.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2008 ACM 1550-4832/2008/04-ART8 $5.00. DOI 10.1145/1350763.1350768 http://doi.acm.org/10.1145/1350763.1350768

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation; D.2.2 [**Software Engineering**]: Design Tools and Techniques; B.6.3 [**Logic Design**]: Design Aids

General Terms: Design, Verification

Additional Key Words and Phrases: Nanocomputing, cellular array, model checking, symbolic techniques

ACM Reference Format:

Carmona, J., Cortadella, J., Takada, Y., and Peper, F. 2008. Formal methods for the analysis and synthesis of nanometer-scale cellular arrays. ACM J. Emerg. Technol. Comput. Syst. 4, 2, Article 8 (April 2008), 27 pages. DOI = 10.1145/1350763.1350768 http://doi.acm.org/10.1145/1350763.1350768

1. INTRODUCTION

The revolutionary density increases of integrated circuits over the last decades have resulted in powerful computers, but it is expected [ITRS 2005; Bourianoff 2003; Likharev and Strukov 2005] that this progress may level off in the coming decade, sparking new research on alternative technologies and architectures. Much of such research has focused on extending current designs—mostly based on the gating of electrical current by transistors—towards realizations by novel materials, such as carbon nanotubes [Avouris et al. 2003]. Increasing attention has also gone to alternative devices that hold promise beyond VLSI, like single electronics [Ono et al. 2005], molecular electronics [Joachim et al. 2000], and spintronics [Sarma et al. 2000], as well as to alternative operating principles of devices and circuits. As hinted by Richard Feynman in his famous 1959 address to the American Physical Society, there is indeed plenty of room at nanometer scales to use different physical mechanisms for computation.

Manufacturing nanometer scale structures is also important for realizing these novel computational mechanisms, as pointed out by Feynman. Many of the top-down methods in use today to fabricate circuits—imprinting a design through optical lithography via a mask on a wafer—are unlikely to be stretched to feature sizes below 10 nm, leaving only bottom-up approaches, like molecular self-assembly, a viable option. Such approaches, however, impose strict limitations on designs, which will likely be either random or (semi-)regular [Stan et al. 2003]. Unlike randomness of design, which needs extensive post-fabrication reconfiguration, (semi-)regular structures tend to be better suited to conduct computations. Cellular arrays are an interesting candidate in this context, since they illustrate how complex (universal) computation can be accomplished through the integrated operations of very simple cells, which are nothing more than finite automata. It is thus no surprise that cellular arrays have attracted increasing attention for use as nanocomputer architectures [Biafore 1994; Durbeck and Macias 2001; Beckett and Jennings 2002; Peper et al. 2003, 2004]. Asynchronous timing in such cellular arrays have also been proposed recently [Adachi et al. 2004; Adachi et al. 2004; Lee et al. 2003, 2005; Peper et al. 2003, 2004]. Apart from offering ways to reduce energy consumption through a natural implementation of clock gating, these asynchronous models also offer compatibility to the random nature of phenomena

on nanometer scales, which are governed by various interactions between molecules.

Molecular nanocascades are one example of such systems, and since circuits based on them have actually been realized [Heinrich et al. 2002; Eigler et al. 2004], they play a central role in this paper. They can be intuitively described as rows of toppling dominoes at nanometer scales, which can be configured such that the resulting chain reactions resemble the behavior of computational elements, like wires, gates, etc.

This kind of nanocomputing systems can be realized on extremely small scales, and they signify a radically new approach to nanometer-scale logic. Though it is possible to analyze the interactions between the molecules in molecule cascades, and derive empirical rules for them, it is still a sizeable task, given these rules, to design configurations suitable for logic operations, especially if the number of molecules and the sizes of configurations increase. The same holds for nanocascades and cellular arrays in general. It is thus helpful to have systematic methods to make a selection of promising configurations, before realizing them experimentally. Preferably, such methods are run on computers because of the huge number of possibilities in which such systems can be configured. Even computer-directed search through the huge number of configurations, however, eventually has to face the reality of combinatorial explosions, due to which many reasonably sized configurations are out of reach.

Since the late sixties, computer scientists have been concerned about designing computer systems and proving their correctness [Hoare 1969]. Proving correctness of a system requires examination of its state space, a difficult task often plagued by exponential combinatorial explosions, which prevent straightforward methods like exhaustive enumeration. Abstraction techniques—facilitating the analysis of complex systems [Cousot and Cousot 1977]—allow designers to model the behavior of systems by hiding irrelevant details and instead focusing on analyzing only those characteristics that have some impact on the properties to be verified. Among such techniques is *temporal logic* [Pnueli 1977], which has become a very convenient formalism to specify properties of systems characterized by a dynamic evolution in time, like computer systems. It allows the specification of statements like *"something will always be true," "something will eventually happen"* or *"something will hold until something else happens."* Temporal logic has become the main specification formalism for *model checking* [Clarke and Emerson 1981; Queille and Sifakis 1981], which is used to prove that a model holds in all the states of a system, and which has become the working horse in formal verification, especially in hardware systems, since it can be fully automated. Like many other techniques, however, model checking does suffer from the state explosion problem, and it was not until the appearance of *Binary Decision Diagrams (BDDs)* [Bryant 1986] that its full potential started to be appreciated in formal methods. Novel techniques such as SAT [Moskewicz et al. 2001] have widened the application of model checking to real life problems. Unfortunately, both approaches suffer from the state explosion problem. BDDs are acyclic graphs that can represent Boolean formulae efficiently; they can represent and manipulate large sets of

states in an implicit way, that is, without explicitly enumerating them [Burch et al. 1990].

This article proposes a methodology to analyze, synthesize, and verify cellular array-like systems whose behavior is determined by interactions at atomic scales.[1] The methodology combines some of the BDD-based techniques to deal with the complexity of such kinds of system. Only the relevant information is used to model a system: the presence or absence of molecules, atoms, electrons, or tokens in specific locations is modeled by Boolean variables. The evolution of the system is modeled as a transition system in which the changes of state are the result of the local interactions. The expected behavior of a system is specified by temporal logic formulae. Moreover, BDDs are used to represent the reachable states of the system symbolically. Finally, model-checking techniques are used to algorithmically derive computational structures that behave as logic gates.

The techniques presented in this article are meant to be used as a computer aid for the synthesis and/or verification of cellular array-like systems. Though based on computationally expensive algorithms, the proposed techniques can be used in the synthesis part for deriving a universal (and optimized) library of nanometer-scale components that will form a basis for building—possibly with other techniques—more complex circuits.

The main contributions of the article are applicable to nanocascades and cellular arrays in general, but a particular example based on the CO-molecule nanocascades in Heinrich et al. [2002] will be used in the article to present the methodology. Section 2 intuitively describes the underlying physics behind CO-molecule nanocascades. Section 3 introduces the background required to formally model nanocascade systems. Sections 4 and 5 presents a generally applicable methodology to derive configurations that implement the behavior of combinatorial logic gates. The proposed techniques are applied in Section 6 to derive a library of logic gates based on the molecule cascades. Section 7 shows that the proposed methodology is not only applicable to simple cellular arrays involving nanocascades, but also to (more complicated) asynchronous cellular arrays in general. The method can be used in a similar way as in Sections 3 to 5 to find configurations of cells with certain functionalities, but we will rather apply it to verify the correctness of a configuration in an asynchronous cellular array, given its transition rules. Though less complex a task than searching for functional configurations, verification is still a nontrivial effort, especially when it concerns asynchronously timed systems.

1.1 Related Work

The use of formal methods to derive valid structures has been addressed in the literature by some authors in different contexts. In Devadas [1989], a BDD-based Boolean satisfiability approach was used to encode the problem of finding an optimal layout for integrated circuits. Another remarkable approach is presented in Ling et al. [2005], where Field Programmable Gate Arrays can be synthesized by solving Quantified Boolean Formulae.

[1]This article is an extension of the work presented in Carmona et al. [2006].

The approaches presented in Devadas [1989] and Ling et al. [2005] translate the problem into a satisfiability check. For instance, in Devadas [1989], channel routing is translated into a set of constraints that must be fulfilled in order to guarantee that a channel can be routed in D tracks. Once the problem has been solved, the routing must be done without taking into account the behavior of the underlying circuit. The current paper, on the other hand, includes a *time dimension* in the analysis: the dynamics of the computed objects are crucial for whether a solution is accepted or not. Hence the main difference between our approach and the other approaches referred to above is the handling of the dynamics into the search for solutions. We handle this dynamic behavior by transforming the problem into a model checking problem over a CTL formula, where traversal techniques can be used to search for solutions. Given its nature, the approach presented in this paper can in principle be translated into a satisfiability check.

Formal methods have not been used before—as far as we know—to synthesize logic structures in nanometer-scale cellular arrays. Due to their regularity, such cellular arrays are not only suitable for the synthesis of a standard library of gates by formal methods, they also potentially allow the use of bottom-up manufacturing methods, based on techniques like molecular self-assembly. The two computational models used in this paper, nanocascades and triangular self-timed cellular automata, both have a cellular array structure, so they are amenable to these advantages.

2. MOLECULAR SYSTEMS

2.1 Nanocascades

Nanocascades are cascades of nanometer-scale entities (particles or molecules) that move from high-energy to low-energy states through a chain reaction in which one entity's move triggers that of the next entity, like a row of toppling dominoes. Consisting of discrete nanometer-scale entities, they occupy a discrete state space. The discrete nature of the state space is rooted in the shape of the energy landscape corresponding to the system. This landscape consists of potential wells arranged in, for example, a regular pattern, like an egg carton. The entities, which we shall call *tokens*, can be thought of as being trapped in certain wells in this energy landscape, thus forming a *configuration*. When an amount of energy is injected in a token that is sufficient to "climb the wall" around the corresponding well, the token may escape toward another well. Alternatively, even without the injection of energy, a token may tunnel through the wall, due to the quantum-mechanical nature of the system; in this case the move of a token between wells is probabilistic. The presence of a token in a well may deform the energy landscape, as a result of which some walls may locally have a lowered profile. This may influence the ease by which a token moves between wells, due to its energy becoming sufficiently high to surpass the wall, or due to the probability of tunneling between two wells along a lowered wall becoming higher. When a token moves from one well to another, the configuration of the system changes, that is, it undergoes a *state transition*. The rules

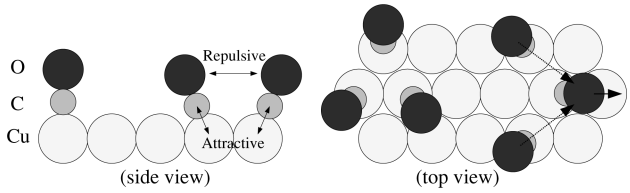


Fig. 1. Tilting of CO molecules as a result of mutually repulsive forces.

describing under which conditions a state transition takes place are called the *transition rules* of the system. In a nanocascade, configurations are designed such that one state transition triggers the next state transition, which in turn triggers a third state transition, and so on. Typically, the tokens in a nanocascade are in a metastable configuration that can be triggered to cascade into a lower energy configuration by transitions of other tokens. This behavior can be exploited, for example, to transmit signals over a grid of atoms, or even to conduct logical operations, like AND and OR.

2.2 Nanocascades Based on Molecular Interactions

Molecule cascades are nanocascades in which interactions take place between molecules. Researchers have succeeded [Heinrich et al. 2002] to create molecule cascades of Carbon Monoxide (CO) molecules arranged on a copper (Cu) surface, in which the grid points are in the 111-plane of the Cu crystal structure. Denoted as a Cu(111), this surface corresponds to a triangular grid. Carbon monoxide (CO) molecules, placed on top of this surface, will bond to the grid points, with the carbon atoms oriented towards the surface. The interaction of CO molecules with metallic surfaces has been studied extensively, for example for Pd(110) [Kato et al. 2000], Ni(100), Pd(100), and Cu(100) [Uvdal et al. 1988]. Apart from the bonding of CO molecules to a Cu surface, there are also interactions *between* CO molecules, through repulsive and attractive forces, like Pauli repulsion, Born repulsion, van der Waals attraction, and electrostatic forces. Due to repulsion, CO molecules tend to avoid being in nearest neighbor sites. Once two CO molecules occupy neighboring sites, however, the carbon-copper bonding is strong enough to keep them in place, though they will tilt away from each other to maximize their distance (Figure 1).

Tilting of crowded CO molecules on metallic surfaces can be modeled as a spring lattice model, whereby each CO molecule is regarded as a rigid mass bound with springs to a flat surface and to other neighboring CO molecules [Kato et al. 2000]. Though this model neglects many parameters corresponding to the detailed lattice structure, it is intuitive and serves our purposes well. When distances between CO molecules increase, attractive forces like van der Waals forces become dominant, albeit, unlike with springs, van der Waals forces decrease quickly with increasing distances, so the spring model has limited value in this case.

In many configurations of CO molecules, the carbon-copper bonding is strong enough to keep CO molecules anchored to Cu grid points, that is, as when we have an isolated CO molecule, a pair of CO molecules at their nearest

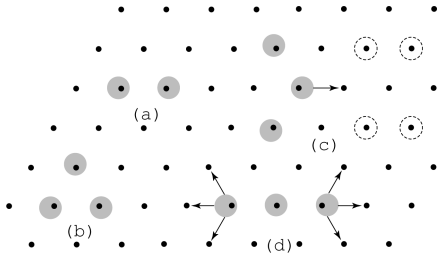


Fig. 2. Stable and unstable configurations of CO molecules. Tilting of a molecule is indicated by a displacement with respect to its grid point. (a) Two CO molecules in their nearest neighbor configuration are stable, as well as (b) three molecules. (c) The *chevron* configuration is unstable and the central molecule will eventually tunnel to the right. This process can be impeded by placing molecules at the two left-most dashed circles at distance 1 from the destination of the central molecule. On the other hand, the process is accelerated by moving the two molecules to the two dashed circles at the right, since those molecules weakly attract the center molecule. (d) Three molecules placed on a line are very instable, as a result of which one of the side molecules will move away quickly from the central molecule, to a grid point indicated by the arrows.

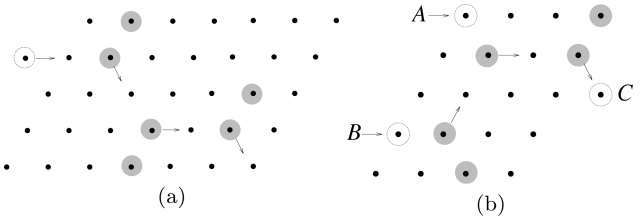


Fig. 3. (a) Linked chevron cascade that can be used to propagate a signal from the upper left to the lower right, (b) Operation $C = A \wedge B$ implemented by a chevron cascade.

(distance 1) grid points (Figure 2(a)), or a triangle of nearest CO molecules (Figure 2(b)). In some configurations, however, repulsion becomes so strong that it cannot be compensated for by tilting of the CO molecules. In that case, a CO molecule tunnel to a neighboring grid point, a phenomenon that is called *hopping* in Heinrich et al. [2002]. A typical example is the *chevron* configuration in Figure 2(c), where the central molecule has tilted to the extent that it will hop one grid point to the right. Figure 2(d) shows another example where hopping takes place. In this case, the left and right molecules are tilted to a great extent, since the central molecule exerts strong repulsive forces on them due to its upright position. As a result, one of the left or right molecules will hop away from the central molecule, possibly biased in an upward or downward direction.

Hopping of CO molecules has been exploited by researchers from IBM to create cascading systems of wires and gates [Heinrich et al. 2002; Eigler et al. 2004]. Figure 3(a) shows a configuration of a linked chevron cascade, in which the hopping of the left-most molecule is carried over to the next, and so on. Hopping of a chevron cascade can also be used to construct logic gates. Figure 3(b) shows an AND-gate based on the chevron interaction. These elements have been ingeniously combined into a 3-bit sorter circuit, of which a fascinating video is available online [IBM 2002]. Though the molecule cascades in Heinrich et al. [2002] are still far from practical applications, their

disadvantages—like an operating temperature below 5K, the need to set up molecules by a Scanning Tunneling Microscope (STM), and a setup and operating time in the order of hours—can in principle be overcome by using more suitable molecules, and by the ongoing technological progress.

As pointed out in Heinrich et al. [2002], the above configurations for signal propagation and the AND-gate are for *one-time computation* only, since molecules that are energetically unstable tend to hop to lower energy states, but not back. Once used, a configuration needs to be reset in its original state to conduct an operation again. A manual reset mechanism, using the tip of an STM, is employed in Heinrich et al. [2002]. No fundamental arguments are currently known to preclude the existence of a reset mechanism inherent in a molecular system [Eigler et al. 2004]. Obviously, a reset mechanism—if it exists—will necessarily consume energy for its operation, since it has to bring the system back into a higher energy state.

3. ABSTRACT MODEL FOR MOLECULAR SYSTEMS

We introduce an abstract model to represent the behavior of the CO-molecule-based cascades introduced in Heinrich et al. [2002]. Development of the model for nanocascades on different grids can be done along the same lines.

3.1 Boolean Functions

We aim to implement Boolean functions by molecular systems. A *Boolean function* F is a mapping $F : \mathbb{B}^n \mapsto \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$. Boolean functions are composed of Boolean variables and operators. We use $\neg$, $\wedge$, and $\vee$ to denote the complement, conjunction, and disjunction operators, respectively.[2] We also use implication operators

$$a \Rightarrow b \equiv \neg a \vee b$$
$$a \Leftrightarrow b \equiv (a \wedge b) \vee (\neg a \wedge \neg b)$$

and the conjunction ($\wedge$) and disjunction ($\vee$) quantifiers. A literal is a variable (x_i) or its complement ($\overline{x}_i$). A minterm of an n-variable Boolean function $F(x_1, \ldots, x_n)$, is a product term with n literals from different variables.

Given a function $F(x_1, \ldots, x_n)$, the positive and negative *cofactors* of F with respect to x_i are defined as

$$F_{x_i} = F(x_1, \ldots, x_i = 1, \ldots, x_n)$$
$$F_{\overline{x}_i} = F(x_1, \ldots, x_i = 0, \ldots, x_n)$$

We will also use the *existential* and *universal abstraction* of a function $F(X)$ with respect to x_i:

$$\exists x_i.F = F_{x_i} \vee F_{\overline{x}_i}, \qquad \forall x_i.F = F_{x_i} \wedge F_{\overline{x}_i}$$

Cofactors and abstractions can be naturally extended from one variable to sets of variables.

[2]For simplicity, we will also use $\bar{}$ and $\cdot$ to denote the complement and conjunction, respectively.

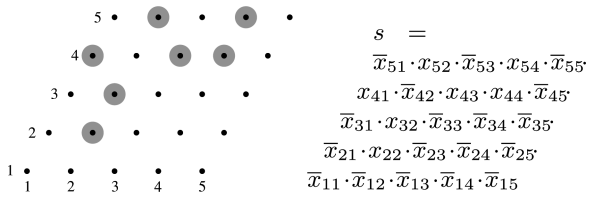


Fig. 4. State of a molecular system.

3.2 Transition Systems

A *transition system* (TS) is a triple $\mathcal{M} = \langle S, T, s_0 \rangle$, where S is a set of states, $T \subseteq S \times S$ is a set of transitions and $s_0 \in S$ is the initial state. A transition (s, s') is also represented by $s \rightarrow s'$. A *path* of length n in $\mathcal{M}$ is a sequence of transitions $s \rightarrow s_1, s_1 \rightarrow s_2, \ldots, s_{n-1} \rightarrow s_n$, and is represented by $s \stackrel{n}{\rightarrow} s_n$. A path of any length (including zero length) is represented by $s \stackrel{*}{\rightarrow} s'$. A state s in $\mathcal{M}$ is said to be *reachable* if there is a path $s_0 \stackrel{*}{\rightarrow} s$.

Molecular systems can be modeled as transition systems in which states represent molecular configurations and transitions represent molecule hops.

3.3 Configurations of Molecules (States)

A molecular system evolves on a finite grid of points in which each point can potentially hold a molecule. The presence or absence of a molecule at each point is modeled by a Boolean variable. A configuration (state) of a system with n grid points can be represented by a *minterm* of an n-variable Boolean function.

Figure 4 depicts a 5×5-grid system. The state of point (i, j) in row i and column j is modeled by variable x_{ij}. The state s of the system is represented by a minterm in which the literals x_{ij} and $\overline{x}_{ij}$ denote the presence and absence, respectively, of a molecule in the point (i, j). In the following sections X and Y will represent sets of state variables.

3.4 Hopping of Molecules (Transitions)

The molecule-hopping mechanism illustrated in Figure 3 is modeled by a set of local transition rules in the transition system. These rules can be specified as transition relations in the Boolean domain. Let state s be represented by an n-variable minterm

$$s(X) = \bigwedge_{i,j} \hat{x}_{ij}, \qquad \hat{x}_{ij} \in \{\overline{x}_{ij}, x_{ij}\},$$

then a transition $s \rightarrow s'$ can be represented by a $2n$-variable minterm $s(X) \cdot s'(Y)$, where $s(X)$ and $s'(Y)$ represent the states s and s' before and after the transition, respectively.

A transition rule R is characterized by a sub-configuration with an unstable molecule (*enabling* condition) that can change its location through a molecule hop (*firing*). The configuration in Figure 5(a) is unstable unless there are two molecules at the points holding the dotted circles. The configuration in Figure 5(b) is still unstable, but the one in Figure 5(c) is stable, since the

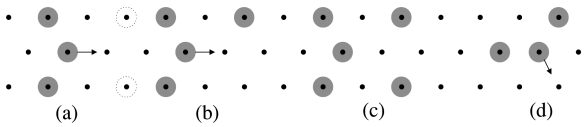


Fig. 5. Possible transitions of configurations.

repulsive force of the two molecules at the right prevent the molecule in the middle to hop.

If the unstable molecule is in location (i, j), the rule corresponding to the conefigurations in Figures 5(a)–5(c) can be characterized by the following Boolean functions:

$$\begin{aligned}
\textsc{Enabling}_{i,j}(X) \;=\; & x_{i+1,j-1} \cdot x_{i,j} \cdot x_{i-1,j} \cdot \\
& \overline{x}_{i,j+1} \cdot \overline{x}_{i+1,j} \cdot \overline{x}_{i-1,j+1} \cdot \\
& (\overline{x}_{i+1,j+1} \vee \overline{x}_{i-1,j+2}) \\
\textsc{Firing}_{i,j}(X,Y) \;=\; & (x_{i,j} \cdot \overline{y}_{i,j}) \cdot (\overline{x}_{i,j+1} \cdot y_{i,j+1}) \cdot \\
& \bigwedge_{(k,l) \notin \{(i,j),(i,j+1)\}} (x_{k,l} \Leftrightarrow y_{k,l})
\end{aligned}$$

The interpretation of these equations is as follows. The enabling condition models the configurations in which the molecule at (i, j) is unstable and willing to hop to $(i, j+1)$. This occurs when the three shadowed locations in Figure 5(a) have a molecule and the three locations next to them are empty. Moreover, it is also required that at least one of the dotted locations is free (represented by the disjunction in the formula). The rule is enabled in a state s if

$$s(X) \implies \textsc{Enabling}_{i,j}(X).$$

If the rule is enabled in s, a new state s' can be reached. The relation between s and s' is modeled by the FIRING predicate. It indicates that all the locations (k, l) have the same value $(x_{k,l} \Leftrightarrow y_{k,l})$ except for locations (i, j) and $(i, j+1)$ in which the configuration changes. Formally, s' is reachable from s in one hop using this rule if

$$s(X) \cdot s'(Y) \implies \textsc{Firing}_{i,j}(X, Y).$$

Every *abstract* rule can have different instances when applied to different grid points using different rotations. In an $n \times m$ grid, the rule depicted in Figure 5 can have up to $6nm$ instances when applied to every grid point in the six different rotations (multiples of 60°). Figure 5(d) illustrates one of the possible rotations of the rule.

4. MODEL CHECKING OF MOLECULAR SYSTEMS

The problem of finding logic gates in a molecular grid space can be reduced to a *model checking* problem: finding initial molecular configurations that satisfy a set of properties. Model checking is a discipline that has been intensively studied, and different tools and techniques have been proposed for several areas of applications in which formal verification isrequired [Clarke et al. 1999]. This

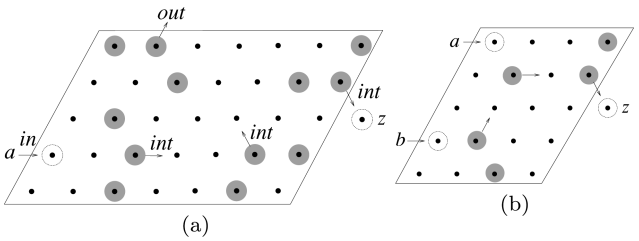


Fig. 6. (a) Classification of the rules, (b) Example of AND gate.

article employs Computational Tree Logic (CTL) [Clarke et al. 1986] to describe the properties that characterize logic gates. For the reader unfamiliar with CTL we have added short explanations of some temporal operators, but a more detailed understanding will require knowledge of CTL syntax and semantics.

Molecular systems are inherently concurrent, since every configuration may have multiple unstable molecules that can hop independently. Concurrency is one of the main causes of combinatorial explosions of states in transition systems. As typically done with model checking techniques, the combinatorial explosion of configurations is tackled by representing sets of states symbolically in a more condensed form; we have used Binary Decision Diagrams [Bryant 1986] to this end.

4.1 Finding Gates

Assume gates have two inputs, a and b, and one output z. The extension to larger gates is straightforward. Consider a molecular $n \times m$ grid and a set of rules $\mathcal{R}$ describing the dynamics of the underlying system. $\mathcal{R}$ can be partitioned into three categories (see Figure 6(a)):

(1) $\mathcal{R}_{in}$, the set of rules to insert molecules from the environment into specific locations of the grid,
(2) $\mathcal{R}_{out}$, the set of rules to move molecules out of the grid to locations different from z, and
(3) $\mathcal{R}_{int}$, the set of rules to move molecules within the grid.

Figure 6(b) shows an example of an AND gate based on a molecular system that obeys the rules described in Figure 5: the reader can verify that the initial configuration $(\overline{x}_{1,1} \cdot \overline{x}_{1,2} \cdot x_{1,3} \cdot \cdots \overline{x}_{5,3} \cdot x_{5,4})$ ensures the desired behavior. Our goal is to find those initial configurations automatically by satisfying a CTL formula.

Stable Configurations. The initial configurations in the *absence* of molecules at the input locations should be stable, but in the *presence* of these molecules they should become enabled. The CTL formula STABLE describes those configurations as follows:

$$\textsc{Stable}(X) = \bigwedge_{r \in \mathcal{R} \setminus \mathcal{R}_{in}} \neg \textsc{Enabling}_r(X).$$

No External Interaction. We need to avoid configurations where the molecules move out of the grid, except for the particular output locations, like the one depicted in Figure 6(a) with the label z. The formula GOOD_INTERFACE(X) characterizes the configurations that avoid this situation in any reachable state of the system:

$$\text{GOOD\_INTERFACE}(X) = \bigwedge_{r \in \mathcal{R}_{out}} AG(\neg \text{ENABLING}_r(X)).$$

The $AG(p)$ operator indicates that property p globally holds (G) for all paths (A) reachable from the initial state. The AG operator is typically used to specify *invariants* of the system.

Predictable Behavior. In any molecular system, there can be particular configurations for which predicting the behavior of the system is extremely difficult.[3] To avoid visiting such configurations, the analysis is reduced to few specific patterns whose behavior be predicted either analytically or experimentally (e.g., using an STM, as in Heinrich et al. [2002]). For this reason, every system is assumed to have a set $\mathcal{U} \subseteq \mathbb{B}^n$ of undesired configurations (see next section for an example). One should avoid initial configurations that include or lead to these undesired configurations. The formula SAFE(X) ensures this:

$$\text{SAFE}(X) = \bigwedge_{c \in \mathcal{U}} AG(\neg c(X)).$$

Logic Behavior. The functionality of any logic gate can be expressed by a truth table. A gate evaluates to *true* when the following predicate holds:

$$\text{GATE\_TRUE}(X) = AF\ AG(z).$$

The $AF(p)$ temporal operator indicates that property p will eventually hold in the future (F) for any path (A) in the transition system. When combined with $AG(z)$, the predicate indicates that every trace in the system will eventually reach a state in which the output location z holds a molecule, and that molecule will be kept stable in the future.

Similarly, a gate evaluates *false* when the output location z does not hold a molecule, that is,

$$\text{GATE\_FALSE}(X) = AF\ AG(\neg z).$$

The behavior for any input assignment can be described with the previous predicates. For instance, to model the AND gate behavior for the (1,1) input assignment, the following CTL formula is used:

$$(a \wedge b) \Rightarrow \text{GATE\_TRUE}(X).$$

[3]Calculating the exact interaction among n molecules requires the solution of the very complex n-body problem.

A logic gate is represented as the conjunction of the CTL formulas for every row of its truth table. The CTL formula for the AND gate will then be:

$$\begin{aligned}&\textsc{Gate}(X) = \\ &\quad [(\neg a \wedge \neg b) \Rightarrow \textsc{Gate\_False}(X)] \wedge \\ &\quad [(\neg a \wedge b) \Rightarrow \textsc{Gate\_False}(X)] \wedge \\ &\quad [(a \wedge \neg b) \Rightarrow \textsc{Gate\_False}(X)] \wedge \\ &\quad [(a \wedge b) \Rightarrow \textsc{Gate\_True}(X)].\end{aligned}$$

In conclusion, the initial configurations that behave as a particular logic gate are characterized by the following CTL formula:

$$\begin{aligned}\textsc{Confs}(X) = \textsc{Stable}(X) \wedge \textsc{Good\_Interface}(X) \wedge \\ \textsc{Safe}(X) \wedge \textsc{Gate}(X).\end{aligned}$$

Each assignment of X represents a valid configuration for the gate.

5. EFFICIENT REPRESENTATION AND EXPLORATION OF CONFIGURATIONS

The set of all possible valid configurations implementing a gate can be exponentially large in terms of the size of the grid. However, symbolic techniques similar to the ones used for model checking finite systems can also be applied in this context. For those readers unfamiliar with these techniques, a short summary is presented below.

5.1 Binary Decision Diagrams

Binary Decision Diagrams (BDDs) [Bryant 1986] are a reduced form of Decision Trees. The Boolean functions described in Sections 3 and 4 can be efficiently represented and manipulated with BDDs. Besides being canonical, BDDs are often more compact than other representations, such as conjunctive or disjunctive normal forms. In the area of computer-aided design, BDDs have been extensively used.

Stone's representation theorem [Stone 1936] states that *every Boolean algebra is isomorphic to the Boolean algebra of sets*. Based on this important result, the sets of configurations can be represented by Boolean functions and, thus, by BDDs. In particular, a set S can be represented by the Boolean function $S(X)$, where X is a set of variables encoding the elements of the set. The predicate $x \in S$ can be described as $S(x) = 1$ and operations on sets can be expressed as Boolean operations. The set of transitions in a transition system can be represented with a Boolean function $T(X, Y)$, called *transition relation*, where X and Y are disjoint sets of variables representing the present and next state, respectively.

5.2 Symbolic Reachability Analysis

This section describes the classical symbolic algorithm to calculate the reachable states of a finite-state system [Burch et al. 1990; McMillan 1994].

Consider the transition system $\mathcal{M} = \langle S, T, S_0 \rangle$ that models a molecular system in a $n \times m$ grid with S_0 as initial set of molecular configurations, represented

by the BDD $S_0(X)$. Moreover let the BDD $T(X, Y)$ represent the transition relation T. The goal is to compute the set S by traversing the edges in T. In set notation, the set of molecular configurations S_1 reachable in at most one hop from S_0 is

$$S_1 = S_0 \cup \{s' \mid \exists s \in S_0 \wedge (s, s') \in T\}.$$

Given the BDDs $S_0(X)$ and $T(X, Y)$, a BDD representing S_1 can be computed by applying only boolean operations corresponding to the expression above:

$$S_1(Y) = S_0(Y) \vee \exists_{x \in X}[S_0(X) \wedge T(X, Y)],$$

where the existential quantifier indicates quantification over all variables in X, thus representing the reachable states with the variables Y. The general predicate for describing the set of molecular configurations that can be reached in at most $k + 1$ hops is

$$S_{k+1}(Y) = S_k(Y) \vee \exists_{x \in X}[S_k(X) \wedge T(X, Y)]. \tag{1}$$

Since $S_k \subseteq S_{k+1}$ and the number of molecular configurations on an $n \times m$ grid is finite, there exists a k for which $S_{k+1} = S_k$. Therefore the symbolic traversal algorithm iterates until the *least fixed point* is reached.

5.3 Partitioned Transition Relation

The transition relation T must contain all possible molecule hops in the $n \times m$ grid space. The BDD for T can be obtained by the disjunction of the BDDs representing all possible molecule hops, that is,

$$T(X, Y) = \bigvee_{r \in \mathcal{R}} T_r(X, Y),$$

where $T_r(X, Y) = \textsc{Enabling}_r(X) \wedge \textsc{Firing}_r(X, Y)$. The BDD $T(X, Y)$, known as the *monolitic transition relation*, can grow exponentially in terms of the number of disjunctions $T_r(X, Y)$. If the number of transition rules is large, then the size of $T(X, Y)$ can be prohibitive. Alternatively, T can be represented as a *partitioned transition relation* [Burch et al. 1994]. In this way, Equation (1) can be rewritten as follows:

$$S_{k+1}(Y) = S_k(Y) \vee \bigvee_{r \in \mathcal{R}} \exists_{x \in X}[S_k(X) \wedge T_r(X, Y)].$$

Hence, the expensive relational product is only performed on small transition relations, one for of each rule. The relations $T_r(X, Y)$ are stored in a list and the traversal is performed by finding applicable relations in this list at each step. This strategy improves the efficiency of the traversal algorithm significantly.

5.4 Optimization and Verification of Properties

Besides finding correct molecular gates, our approach can also be used to find configurations with specific properties. Examples of properties or cost functions that may be included in the search are listed below.

Minimum Number of Molecules. The gates with the minimum number of molecules can for example be obtained by finding the configuration with the

minimum number of positive literals. If the length of an edge in the BDD is zero if a variable is false and one otherwise, this problem reduces to finding the shortest path in the BDD that characterizes all configurations [Lin and Somenzi 1990].

Minimum Number of Rules. The problem of finding a minimum set of rules that is sufficient to implement a given functionality is also possible in our framework. The decision on whether to use or not to use a given rule can be included into the model by means of an extra Boolean variable that assumes the logic value true if and only if the rule is essential.

Formally, the FIRING predicate introduced in Section 3.4 can be extended to account for the rule used in each molecule hop. In a system with q rules, let $R = \{r_1, \ldots, r_q\}$ be a set of additional Boolean variables, with r_i encoding whether rule i is essential. The set of valid initial configurations for a given gate will be represented with the CTL predicate CONFS(X, R). The search for configurations will then be guided by the aim to minimize the number of these variables with the assignment true.

Fast Gates. Configurations with the shortest *critical path* (number of hops from input to output) represent the fastest way to realize the gate. Because performance is an issue (to sort three bits took one hour in the experiments performed in Heinrich et al. [2002]), optimizing the computation inside the gate is crucial. The symbolic traversal algorithm from Section 4 can be adapted to select configurations that induce fewer steps in the traversal (i.e., need fewer interactions to stabilize), thus leading to faster ways to produce the output of the gate. For example, the leftmost OR gate shown in Figure 8 needs four hops to produce an output when the inputs are a = 0, b = 1. The central OR gate is faster, because only three hops are needed in the worst case.

Multiple-Entry Inputs. There are gates in which input signals can enter the gate through multiple locations. This enhancement can be crucial if errors may occur when transmitting the signal to the gate. Gates accepting multiple-entry locations are fault-tolerant to these variations.

Apart from the symbolic methodology for representation and exploration of configurations presented in this paper, other techniques can also be applicable. For example, we mention *Bounded Model Checking (BMC)*, a SAT-based technique that, combined with knowledge about the worst-case length of the possible traces, has proven to be very powerful in the analysis and verification of complex systems [Biere et al. 1999; Clarke et al. 2001]. For the case of nanocascades, once the grid is known, the number of hops needed in the worst case to transmit a signal from inputs to outputs can be easily estimated and this knowledge can be used to guide the BMC methods to find the configurations. The encoding of the problem for using BMC techniques has to be adapted, however, because there is a quantifier alternation that cannot be easily handled with BMC. This quantifier alternation corresponds to the search for a configuration that is correct for every input assignment. When the number of input variables is low, one can use similar techniques to the ones presented in Ling et al. [2005] to remove the outer-most existential quantifier. The use of

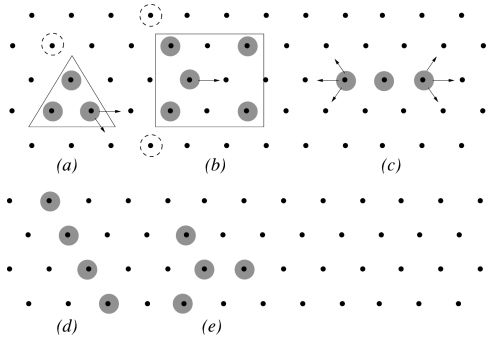


Fig. 7. (a)–(c) Rules derived from IBM, (d)–(e) Prohibited patterns.

BMC techniques can significantly widen the applicability of the methodology presented in this paper.

6. MOLECULE CASCADES: UNIVERSAL LIBRARY OF GATES

In this section, we derive a universal library of gates for the IBM molecular cascades. The library has been automatically obtained through the techniques outlined in the previous sections. Implementation details can be found at the end of the section.

6.1 Rules and unsafe patterns

Apart from the main rules in Figure 2, which are inspired by the heuristic rules in Eigler et al. [2004], there are configurations (Figure 7), including unsafe ones, that need a careful consideration.

Figure 7(a) describes how to destabilize a triangle of three molecules, which, if isolated, would be stable. The presence of an additional molecule near the top of the triangle produces a hop of the bottom right-most molecule. This hop is nondeterministic, since the molecule can hop to two different destinations. In practice, finding gates with deterministic behavior that are based on nondeterministic rules is difficult, since the probability of having a unique observable outcome is drastically reduced when the internal behavior of the gate can diverge. On the other hand, nondeterministic rules may be crucial to implement special gates like arbiters (an example appears in Eigler et al. [2004]). For the sake of completeness, we add a rule corresponding to the behavior in Figure 7(a).

The configuration in Figure 7(b) is stable, but when an additional molecule is placed in one (or both) of the dashed circles, the central molecule will hop to the right. So, we add a rule corresponding to this behavior.

The rule for the configuration of three molecules on a line in Figure 7(c) is highly nondeterministic, as we have seen in Section 2. Even if this rule could be considered legal, the chances of finding a gate using it are extremely small.[4] For this reason, we have declared this rule *unsafe*.

Finally, the two configurations shown in Figures 7(d)–(e) describe situations with highly unpredictable behavior, given the complexity of the interactions

[4]Actually, no gate in [Eigler et al. 2004] uses this rule.

Table I. A Complete Family of Dual-Rail Gates

AND	OR	NOT
$z = x \wedge y$	$z = x \vee y$	$z = \neg x$
$z^t = x^t \wedge y^t$ $z^f = x^f \vee y^f$	$z^t = x^t \vee y^t$ $z^f = x^f \wedge y^f$	$z^t = x^f$ $z^f = x^t$

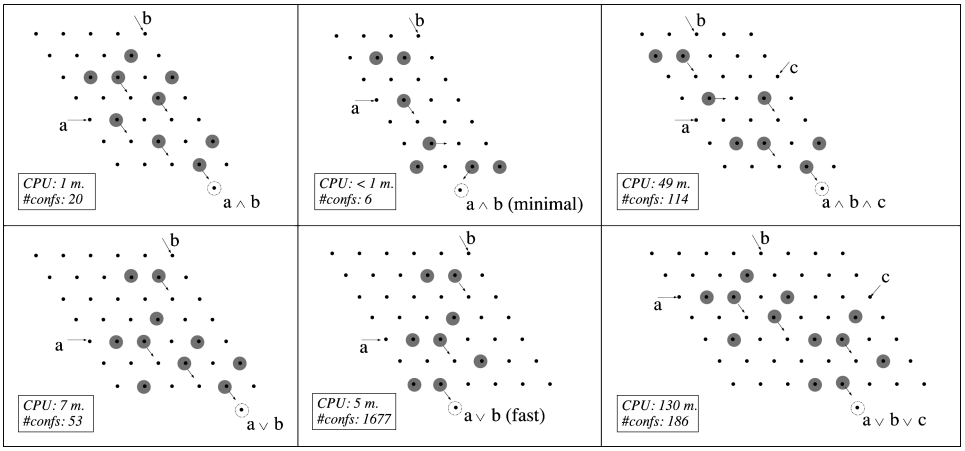


Fig. 8. Library of gates: three implementations for the AND and OR gates.

between the molecules. Accordingly, these configurations have been declared *unsafe* as well.

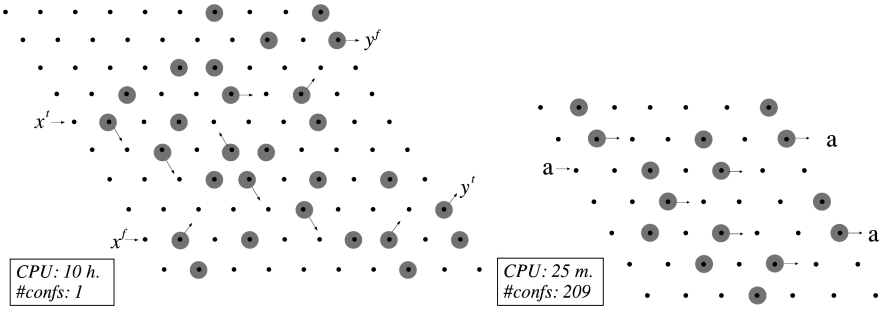
6.2 Complete Library of Dual-Rail Gates

Molecule cascades can implement positive gates in a natural way.[5] Negative gates are difficult to implement, however, since this would require the output molecule to "untopple" when the input molecules topple [Eigler et al. 2004]. Dealing with this problem requires that negative functions are implemented by positive gates, and delay-insensitive codes [Verhoeff 1988] are very helpful to this end. The most popular of these is *dual-rail encoding* [Muller and Bartky 1959], in which every bit x of information is encoded by two signals $x = (x^t, x^f)$. The values $x = 0$ and $x = 1$ are encoded as (0,1) and (1,0), respectively. The combination (0,0) is used as a *spacer* between subsequent signals, and (1,1) is unused. For one-time computation, the spacer is not needed.

Table I describes how dual-rail gates can be implemented using single-rail positive gates. The implementation of a NOT gate is simply done by crossing the x^t and x^f wires.

Figure 8 presents a library with different implementations of AND/OR gates. In the same way as is done for *standard-cell* libraries, grid spaces with a fixed height have been used, whereas widths may vary from one implementation

[5]The Boolean expression of a positive (negative) gate can be represented by a sum-of-products with only positive (negative) literals. For example, AND and OR are positive gates, whereas NOT, NAND and NOR are negative gates.



9.1: A dual-rail NOT gate. 9.2: Fork.

Fig. 9. Gates to complement or fork a signal.

to another.[6] In the first column, two implementations for the AND/OR gates are presented, in which entry and exit points are fixed. CPU times for the computation[7] and the number of valid configurations found are also shown in the figure. Apart from the configuration shown for the AND gate, 19 more configurations are valid. The presented techniques can also be used to obtain negative results: we have found, for example, that it is impossible to lay out an OR gate in a grid with the same size and the same entry/exit points as the grid used for the AND gate (this explains why the OR gate is one unit wider than the AND gate).

In the second column of Figure 8, configurations satisfying a certain property (see Section 5.4) are presented: a configuration with the minimal number of molecules for the AND gate, and a configuration that minimizes the length of the critical path for the OR gate. For finding those optimal configurations, the selection of the optimal exit point is left to the tool. It turns out that the OR gate requires one internal hop less for the input assignment (0,1) than the OR gate of the first column.

Finally, the third column of Figure 8 shows the implementations of a 3-AND gate and a 3-OR gate, the former of which requires no additional area as compared to the 2-AND gate. For these 3-input gates, it was left to the tool to decide the three entry points and the exit point, by means of additional variables in the model to encode the input/output locations. This explains the increased requirements in CPU time.

The configuration in Figure 9 implements a dual-rail NOT-gate. Intuitively, the gate simply crosses the wires x^t and x^f. It can be verified easily, however, that the gate only works correctly when the two wires are mutually exclusive, but this poses no problems in dual-rail encoding, since the combination (1,1) is supposed to never occur. A crossover gate is also presented in Eigler et al. [2004], and it does not work for the (1,1) assignment either. The construction of a complete crossover gate that works correctly for all logic combinations seems

[6]Although not done in the simulations, it is possible to use fixed entry/exit points in order to facilitate the synthesis of gates.

[7]The experiments have been carried out on a desktop with a 1.86 GHz Pentium M 750 processor and 2GB of memory.

Table II. Implementation Details on the Search for Gates

gate	# vars	# rules	# partitions	top BDD	CPU
AND	89	739	2	49072	1 m.
ANDMIN	73	445	1	10130	< 1 m.
AND3	98	748	8	1221241	49 m.
OR	105	1035	8	158038	7 m.
ORFAST	105	1039	8	80999	5 m.
OR3	111	1041	8	3116800	130 m.
FORK	120	1331	8	146346	25 m.
NOT	191	792	64	9438623	10 h.

to require additional considerations about the specific orientations of the O atoms with respect to the second layer of Cu atoms directly beneath the top layer [Heinrich et al. 2002]. Using our systematic approach, we did not find (even though several different grid spaces were explored by the system) such a complete crossover gate, when considerations about orientations of the second layer were left out. This suggests that other physical considerations must be included into our logic model (timing assumptions, orientation with the second-layer, etc.) to find a complete crossover gate.

Finally, a cell to fork a signal is presented in Figure 9. It has the same height as the gates in the library.

Implementation Details. Let us give an insight of the prototype tool created for implementing the techniques described in this paper. The tool has been implemented in C, and uses the CUDD BDD package [Somenzi].

Some BDD optimization strategies have been applied to control the size of the BDDs and to speed-up the search, with different outcomes. BDD partitioning [Sahoo et al. 2004] was used, and in general it induced a significant improvement in the search for valid configurations. In the experiments, the size of the partition (number of portions to represent the initial BDD) was decided from the number of variables of each problem instance. BDD variable reordering was also applied, both dynamic and static, but in general it did not have a strong impact on the BDD sizes. In the experiments, dynamic variable reordering provoked a significant CPU overhead and therefore it was not applied. Hence variable reordering was applied only at the first stages of the search.

Table II summarizes the explorations performed to find the configurations shown in Figures 8,(9.2) and (9.1). Column labeled "# vars" indicate the number of variables needed to encode the problem, which include the present and next state sets of variables needed for the traversal, and the encoding of the input and output locations. Column "# rules" reports the number of rules applicable in the corresponding grid space.[8] Column "# partitions" and "top BDD" report the size of the partition used and the maximum BDD size (in number of nodes) found in the exploration, respectively. Finally, the CPU time for each search is given in the last column.

From the table it can be seen a correlation between the growth of the number of variables of the problem instance and the growth of the BDD sizes and CPU

[8]For the NOT gate, a partial assignment to the initial configuration was done in order to avoid a large set of rules.

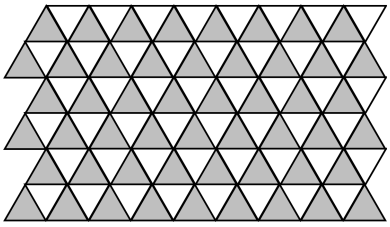


Fig. 10. Cell space of the Triangular Self-Timed Cellular Automaton (TSTCA). Cells are divided in two classes, white and gray cells, each having their own state transition rules.

times. The CPU times reported can be acceptable when the goal is to build optimized libraries for new nanotechnologies like molecule cascades. The use of SAT or BMC techniques can be used instead of the BDD approach to widen the applicability of the methods described in this paper.

7. VERIFICATION OF TRIANGULAR SELF-TIMED CELLULAR AUTOMATA

Apart from being suitable to model nanocascades on molecular grids, the techniques developed thus far also find use in more general models, like cellular automata (CA). In this section we will focus on CA that lack a central clock by which all cells are updated synchronously. Called *asynchronous CA*, such models are attracting increasing interest as architectures for nanocomputers, because of their compatibility with the probabilistic behavior usually found on nanometer scales, and because of their regular structures, which facilitate bottom-up manufacturing methods based on molecular self-assembly. In recent years, computationally universal asynchronous CA have been proposed [Adachi et al. 2004; Lee et al. 2003, 2005; Peper et al. 2003, 2004] that abandon any synchronization between cells other than in local neighborhoods of cells. Though these models tend to be more efficient than asynchronous CA that are based (through simulation [Lee et al. 2004]) on synchronous CA, they are hard to verify—other than by hand or by heuristic methods—due to the unpredictability by which the order of state transitions take place. In this section, we outline how our formal methods can be applied to the verification of a simple asynchronous CA, the *Triangular Self-Timed Cellular Automaton (TSTCA)*. This model is a triangular version of the (square) STCA models in Peper et al. [2003, 2004].

7.1 Description of the Model

The TSTCA assumes cells that are triangular, as in Figure 10, each having one bit along each of it edges, thus making up for a total of three bits per cell. Apart from its possible $2^3 = 8$ states, a cell has also a color, gray or white in our case, determining the transition rules applicable to it. Each cell is able to read its three own bits, as well as the the nearest bit of each of its three neighboring cells (see Figure 11). This total of six bits can also be written to by the cell in accordance with rules that define how they change through transitions. Figure 12 gives examples of such transition rules, separated in rules for white cells and rules for gray cells. Cells will be updated in a random order, but a cell will only

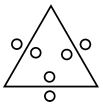


Fig. 11. Cell of the Triangular Self-Timed Cellular Automaton. Each cell has three bits (shown here as small circles) to encode its state. When a bit is in state 1, it is shown in subsequent figures as a black (filled) circle; when it is 0 it is a blank (empty space). A cell can read its three bits and the three directly neighboring bits. It can also write to those six bits in accordance with its transition rules.

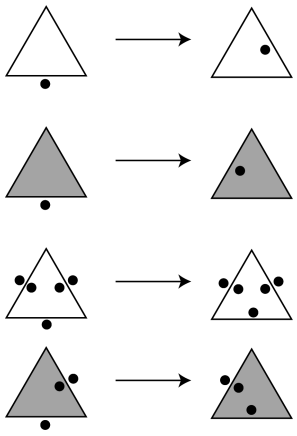


Fig. 12. Transition rules of the Triangular Self-Timed Cellular Automaton. The first two rules are used for the propagation of signals in white resp. gray cells. The combined effect of these rules—turning signals right in white cells and left in gray cells—results in signals propagating along straight paths over the cell space. The third rule reverses a signal entering a white cell to where it came from. In combination with the second rule this results in a hard right turn of the signal. The fourth rule is used as a switch for gray cell, into one of three cyclical states.

be updated if the states of its six corresponding bits match the Left Hand Side (LHS) of at least one transition rule. Furthermore, simultaneous update of two neighboring cells is prohibited, to avoid situations in which the two common bits of these two cells are updated by the two cells at the same time, which would create writing conflicts. Each of the rules in Figure 12 can be applied to any of the three 120-degree rotations that may be assumed by a configuration.

We will study the construction of a configuration of cells having the functionality of a *Conservative Join (CJoin)*, which is explained in Figure 13. Based on the transition rules in Figure 12, this construction is given in Figure 14. The first two transition rules in Figure 12 are used for signal propagation, whereby a signal is supposed to be a single (black) 1-bit at the edge of a cell. If this bit is at the outside of an edge, it will enter the cell via that edge, and when it is at the inside of an edge, it will leave the cell via that edge. The third rule in Figure 12 reverses a signal entering a white cell to where it came from, whereas the fourth rule acts as a switch on the signal. To facilitate understanding of Figure 14 we have included dashed or black arrows, tracing the paths followed by the two signals input to the CJoin at terminals In_1 and In_2 resp., and output at terminals Out_1 and Out_2.

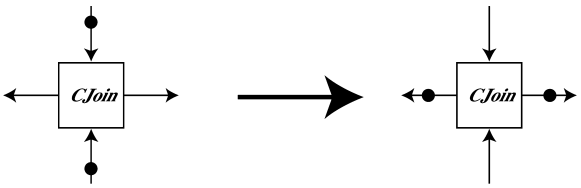


Fig. 13. The Conservative Join (CJoin) module has two input lines and two output lines. If there is one signal input to each of the two input lines (LHS of the transition rule), then the CJoin will produce one signal on each of the two output lines (RHS of the transition rule). When only one of the input lines contains a signal, that signal remains pending until a signal on the other input line appears.

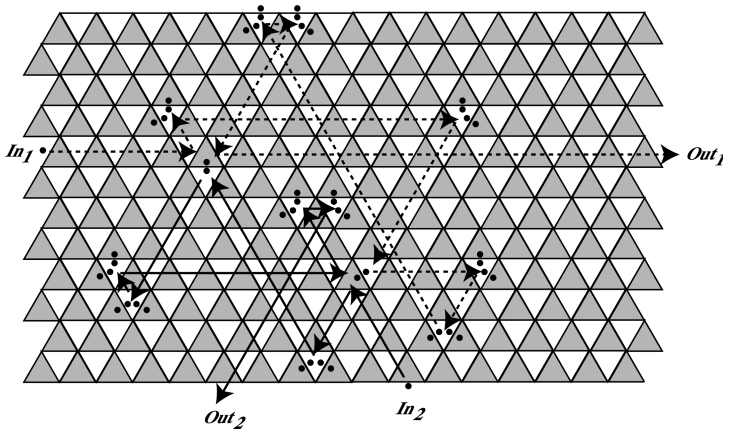


Fig. 14. Implementation of the Conservative Join on the cell space of the Triangular Self-Timed Cellular Automaton. Signals are input at In_1 and In_2, following the paths indicated by the dashed and black arrows, respectively. One of the two switches in the circuit is switched two times by a signal on the dashed path and one time by a signal on the black path, and the other switch the other way around. A total of three switchings per switch is required for each operation of the CJoin module, to assure that the switches return to their initial states after the signals are output to terminals Out_1 and Out_2.

7.2 Formal Verification of Gates in the TSTCA Model

A TSTCA can be modeled in a grid space, as is shown in Figure 15. A cell is encoded with three Boolean variables to represent its 3-bit state. The state variables, together with the variables at the boundaries represent the necessary information to define the rules for a cell (see Figure 15 (a)).

To distinguish between white and gray cells, different orientations are used. Correspondingly, one can define particular rules for a white or gray cell, depending on the orientation on the grid of the variables corresponding to a cell. The particular orientations of the white and gray cells are characterized in Figure 15(b). A grid containing the small TSTCA space within the dashed square in Figure 15(c) is shown in Figure 15(d), where the grid points within the circles correspond to variables in the model, and the other grid points are dummies.

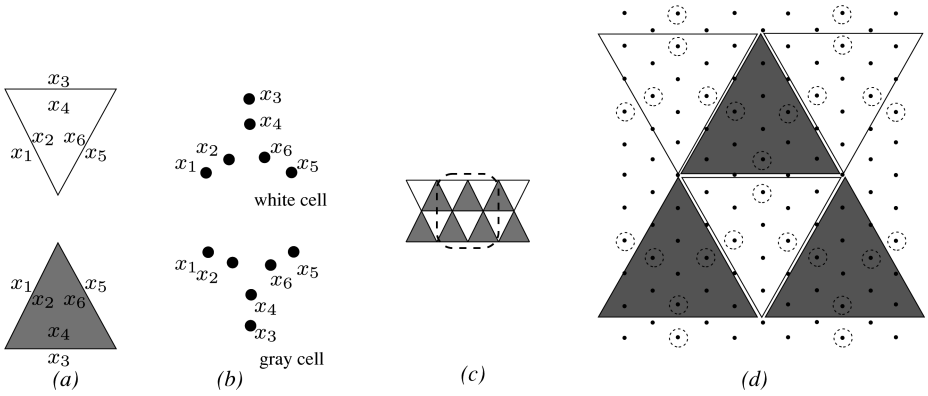


Fig. 15. Mapping of the TSTCA into the grid space.

The rules can be defined for each variable in the grid as follows: depending on a cell's position in the grid, a variable represents part of the internal state of a white or gray cell. For each one of the abstract rules defined in Figure 12, three instances of the rule are created in each one of the cells of that type in the grid space. These instances correspond to the three 120-degree rotations of that rule.

For instance, the rule for signal transmission in the white cell with the variables names defined in Figure 15(b), where the dot arrives through the boundary x_1 is characterized by the following Boolean functions:

$$\textsc{Enabling}(X) = x_1 \cdot \bigwedge_{j \in \{2..6\}} \overline{x}_j$$

$$\textsc{Firing}(X, Y) = (x_1 \Leftrightarrow \overline{y}_1) \cdot (\overline{x}_6 \Leftrightarrow y_6) \cdot \bigwedge_{j \in \{2..5\}} (x_j \Leftrightarrow y_j)$$

Similarly, a transition rule with the dot arriving through x_3 or x_5 will also be created. These are the two possible 120-degree rotations of the rule defined above. Since there are two basic rules for each cell (Figure 12) and three 120-degree rotations are represented for each rule in a cell, we have six rules per cell, so, a grid space with n cells using the abstract rules defined in Figure 12 will have $6n$ rules.

The methods presented in this paper can be used to verify structures in general models like the TSTCA. The verification of a candidate solution is performed by restricting the set $\textsc{Confs}(X)$ defined in Section 4 to a singleton, containing the candidate solution. Then, the techniques presented in Section 4 will efficiently check the conditions on this singleton with low complexity, given that the state space is restricted to the one reached by the only configuration in $\textsc{Confs}(X)$.

Experiments with the TSTCA Model. The CJOIN gate presented in Section 7.1 has been formally verified with the techniques of this paper. Given the configuration of the CJOIN in Figure 14, a big grid space containing the

configuration has been created. The grid space required more than three thousand variables, and more than six thousand rules. Then the variables of the space were assigned to the values defined in the CJOIN configuration in Figure 14, using the mapping defined in Figure 15. Finally applying the techniques on this initial configuration, we proved the CJOIN to be correct. The CPU time required for the verification was approximately 10 minutes.

8. CONCLUSIONS

We have proposed a method to find functional configurations of nanocascades, given the interaction rules governing them. This method is more systematic than the manual method, and allows us to impose specific conditions on the configurations to be synthesized. Proving the nonexistence of gates satisfying certain conditions—a task that is cumbersome by hand—is also possible by the proposed method.

Since the interactions in the nanocascades are represented by logic formulae, there is a limitation to the range of rules that can be handled by the proposed method. Interactions whose description require complex models that include variables with continuous values may be less suitable for our method. Discretization of the variables may offer some solutions in that case, though some nanocascade systems may be more suitable to such an abstraction than others.

In the context of circuits and computation there are still many open issues. An important one has already been mentioned in Section 2: the CO-Cu based molecule cascades are one-time computing structures, since the initial configuration is displaced by the computation. This need not necessarily hold for nanocascades in general: if the elements of a nanocascade are bi-stable, like in Quantum Dot Cellular Automata [Tougaw and Lent 1994], its configuration can be reused after computation. For structures that are non-reusable, however, other mechanisms are necessary to facilitate a return to a pre-computation state. Reversible logic [Bennett 1982] is a straightforward candidate, but further research is required to explore its use in this context. Several authors have invested efforts to reversible logic, though not so much motivated by the need to restore the initial state of a computation, as well as to achieve zero- or reduced-energy computation [Frank 2005].

The techniques presented in this article can also be extended to handle more general models, like asynchronous cellular arrays, as is shown on our case study on the TSTCA model. We focused on the verification of a hand-designed configuration, rather than finding such configurations automatically, since the complexity of the latter task—given the large grid sizes involved—proved too overwhelming in terms of computation times in initial tentative trials we made. Generally, the complexity of the analysis techniques for a model increases with the model's expressive power, and the approach presented in this paper is not free from this problem. The combination of partial knowledge on the solution can help in alleviating the complexity of the analysis. Though the asynchronous cellular array, of which the correctness was verified, is relatively simple, we expect that our verification techniques can also be applied to more complicated asynchronous cellular arrays. Given the complexity of such verification tasks

by hand or heuristic techniques, we may conclude that our techniques are thus a powerful tool in this context.

ACKNOWLEDGMENTS

We would like to thank Dr. Yoshishige Okuno at Advance Soft Inc, Dr. Shukichi Tanaka of the Nano ICT group at the National Institute of Information and Communications Technology (NICT), and Dr. Shiyoshi Yokoyama of the Department of Advanced Device Materials Institute for Materials Chemistry and Engineering at Kyushu University, all in Japan, for their advice and discussions about molecular systems.

REFERENCES

ADACHI, S., LEE, J., AND PEPER, F. 2004. On signals in asynchronous cellular spaces. *IEICE Trans. Inf. Syst. E87-D,* 3, 657–668.

ADACHI, S., PEPER, F., AND LEE, J. 2004. Computation by asynchronously updating cellular automata. *J. Statis. Phys. 114,* 1/2, 261–289.

AVOURIS, P., APPENZELLER, J., MARTEL, R., AND WIND, S. 2003. Carbon nanotube electronics. *Proceedings of the IEEE 91,* 11, 1772–1784.

BECKETT, P. AND JENNINGS, A. 2002. Towards nanocomputer architecture. In *Proceedigs of the 7th Asia-Pacific Computer Systems Architecture Conference* (ACSAC '02), F. Lai and J. Morris (Eds.). Vol. 6.

BENNETT, C. 1982. The thermodynamics of computation—a review. *Int. J. Theoret. Phys. 21,* 12, 905–940.

BIAFORE, M. 1994. Cellular automata for nanometer-scale computation. *Physica D 70*, 415–433.

BIERE, A., CIMATTI, A., CLARKE, E. M., AND ZHU, Y. 1999. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems* (TACAS '99). Springer-Verlag, 193–207.

BOURIANOFF, G. 2003. The future of nanocomputing. *Computer 36,* 8, 44–53.

BRYANT, R. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. 35,* 8, 677–691.

BURCH, J., CLARKE, E., LONG, D., MCMILLAN, K., AND DILL, D. 1994. Symbolic model checking for sequential circuit verification. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst. 13,* 4, 401–424.

BURCH, J., CLARKE, E., MCMILLAN, K., DILL, D., AND HWANG, L. 1990. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1–33.

CARMONA, J., CORTADELLA, J., TAKADA, Y., AND PEPER, F. 2006. From molecular interactions to gates: a systematic approach. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*.

CLARKE, E. AND EMERSON, E. 1981. Synthesis of synchronization skeletons for branching time temporal logic. *Logic of Programs: Workshop*. Lecture Notes in Computer Science, vol. 131.

CLARKE, E., GRUMBERG, O., AND PELED, D. 1999. *Model Checking*. The MIT Press.

CLARKE, E. M., BIERE, A., RAIMI, R., AND ZHU, Y. 2001. Bounded model checking using satisfiability solving. *Formal Meth. Syst. Des. 19,* 1, 7–34.

CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst. 8,* 2, 244–263.

COUSOT, P. AND COUSOT, R. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*. 238–252.

DEVADAS, S. 1989. Optimal layout via boolean satisfiability. In *Proceedings of the International Conference Computer-Aided Design (ICCAD)*. 294–297.

DURBECK, L. AND MACIAS, N. 2001. The cell matrix: An architecture for nanocomputing. *Nanotechnology 12,* 3, 217–230.

EIGLER, D. M., LUTZ, C. P., CROMMIE, M. F., MAHORAN, H. C., HEINRICH, A. J., AND GUPTA, J. A. 2004. Information transport and computation in nanometre-scale structures. *Phil. Trans. R. Soc. Lond. A 362,* 1819, 1135–1147.

FRANK, M. 2005. Introduction to reversible computing: Motivation, progress, and challenges. In *Proceedings of the 2nd Conference on Computing Frontiers (CF '05)*. ACM Press, New York, NY, 385–390.

HEINRICH, A., LUTZ, C., GUPTA, J., AND EIGLER, D. 2002. Molecule cascades. *Science 298*, 1381–1387.

HOARE, C. 1969. An axiomatic basis for computer programming. *Comm. ACM 12*, 576–580.

IBM 2002. IBM scientists build world's smallest operating computing circuits. http://domino.watson.ibm.com/comm/pr.nsf/pages/news.20021024_cascade.html.

ITRS 2005. International Technology Roadmap for Semiconductors.

JOACHIM, C., GIMZEWSKI, J., AND AVIRAM, A. 2000. Electronics using hybrid-molecular and mono-molecular devices. *Nature 408*, 541–548.

KATO, H., OKUYAMA, H., ICHIHARA, S., AND KAWAI, M. 2000. Lateral interactions of CO in the (2×1)p2mg structure on Pd(110): Force constants between tilted CO molecules. *J. Chem. Phys. 112,* 4, 1925–1936.

LEE, J., ADACHI, S., PEPER, F., AND MASHIKO, S. 2005. Delay-insensitive computation in asynchronous cellular automata. *J. Comput. Syst. Sci. 70*, 201–220.

LEE, J., ADACHI, S., PEPER, F., AND MORITA, K. 2003. Embedding universal delay-insensitive circuits in asynchronous cellular spaces. *Fundamenta Informaticae 58,* 3/4, 295–320.

LEE, J., ADACHI, S., PEPER, F., AND MORITA, K. 2004. Asynchronous game of life. *Physica D 194*, 369–384.

LIKHAREV, K. K. AND STRUKOV, D. B. 2005. Cmol: Devices, circuits, and architectures. In *Introduction to Molecular Electronics*. Springer, Berlin, Germany, 447–477.

LIN, B. AND SOMENZI, F. 1990. Minimization of symbolic relations. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 88–91.

LING, A., SINGH, D. P., AND BROWN, S. D. 2005. FPGA logic synthesis using quantified boolean satisfiability. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT)*. 444–450.

MCMILLAN, K. 1994. *Symbolic Model Checking*. Kluwer Academic Press.

MOSKEWICZ, M. W., MADIGAN, C. F., ZHAO, Y., ZHANG, L., AND MALIK, S. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*.

MULLER, D. E. AND BARTKY, W. S. 1959. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*. Harvard University Press, 204–243.

ONO, Y., FUJIWARA, A., NISHIGUCHI, K., INOKAWA, H., AND TAKAHASHI, Y. 2005. Manipulation and detection of single electrons for future information processing. *J. Appl. Phys. 97*, 031101–1–031101–19.

PEPER, F., LEE, J., ABO, F., ISOKAWA, T., ADACHI, S., MATSUI, N., AND MASHIKO, S. 2004. Fault-Tolerance in Nanocomputers: A cellular array approach. *IEEE Trans. Nanotech. 3,* 1, 187–201.

PEPER, F., LEE, J., ADACHI, S., AND MASHIKO, S. 2003. Laying out circuits on asynchronous cellular arrays: A step towards feasible nanocomputers? *Nanotech. 14,* 4, 469–485.

PNUELI, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundation of Computer Science*. 46–57.

QUEILLE, J. AND SIFAKIS, J. 1981. Specification and verification of concurrent systems in Cesar. In *Proceedings of the 5th International Symp. on Programming*. Vol. 137. Lecture Notes in Computer Science, 337–351.

SAHOO, D., IYER, S. K., JAIN, J., STANGIER, C., NARAYAN, A., DILL, D. L., AND EMERSON, E. A. 2004. A partitioning methodology for bdd-based verification. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, A. J. Hu and A. K. Martin, Eds. Lecture Notes in Computer Science, vol. 3312. Springer, 399–413.

SARMA, S. D., FABIAN, J., HU, X., AND ZUTIC, I. 2000. Issues, concepts, and challenges in spintronics. In *Proceedings of the 58th IEEE Device Research Conference (DRC)*. IEEE, 95–98.

SOMENZI, F. CUDD: CU decision diagram package. http://vlsi.colorado.edu/ fabio/CUDD/.

STAN, M., FRANZON, P., GOLDSTEIN, S., LACH, J., AND ZIEGLER, M. 2003. Molecular electronics: From devices and interconnect to circuits and architecture. In *Proceedings of the IEEE 91,* 11, 1940–1957.

STONE, M. 1936. The representation theorem for Boolean algebras. *Trans. Amer. Math. Soc. 40*, 37–111.

TOUGAW, P. D. AND LENT, C. S. 1994. Logical devices implemented using quantum cellular-automata. *J. Appl. Phys. 75*, 1818–1825.

UVDAL, P., KARLSSON, P.-A., NYBERG, C., AND ANDERSSON, S. 1988. On the structure of dense CO overlayers. *Surfa. Sci. 202,* 1–2, 167–182.

VERHOEFF, T. 1988. Delay-insensitive codes—an overview. *Distrib. Comput. 3,* 1, 1–8.

Received October 2007; accepted December 2007