From molecular interactions to gates: a systematic approach

Josep Carmona Jordi Cortadella
Universitat Politècnica Catalunya
Barcelona, Spain

Yousuke Takada
Univ. Hyogo
Hyogo, Japan

Ferdinand Peper
NICT
Kobe, Japan

ABSTRACT

The continuous minituarization of integrated circuits may reach atomic scales in a couple of decades. Some researchers have already built simple computation engines by manipulating individual atoms on metal surfaces. This paper presents a systematic approach to automate the design of logic gates using molecule cascades. Temporal logic is used to characterize molecular interactions and specify the behavior of logic gates. Model-checking techniques are used for the exploration of structures behaviorally equivalent to the logic gates. As an example, a complete library of combinational logic gates has been designed using a particular molecular system. This new approach provides a methodology to bridge the gap between physical chemists and computer scientists in seeking computational structures at atomic scales.

Categories and Subject Descriptors: B.6.3 [Hardware]: Logic Design - Design Aids.
General Terms: Algorithms, Design
Keywords: Nanocomputing, formal methods, nanocascades

1. INTRODUCTION

Integrated circuits have experienced revolutionary density increases over the last decades, resulting in powerful computers, but it is expected [3, 13, 18] that this progress may level off in a decade, renewing interest in different technologies and architectures. Much research has focused on extending current designs—based on the gating of electrical current by transistors—towards realizations by novel materials, like carbon nanotubes [1]. Different physical mechanisms to compute on molecular and atomic scales, however, may just as well be possible, and even be more efficient, as hinted by Richard Feynman in his famous 1959 address to the American Physical Society. Single electronics [21], molecular electronics [14], and spintronics [23] may serve as examples that hold promise beyond silicon. There is also increasing interest in alternative circuit architectures, including those based on cellular arrays [8, 17, 22].

1.1 Molecule cascades

Nanocascades are systems of particles (or molecules) arranged such that the interaction of one particle (or a set of particles) with a second particle (or a set thereof) triggers the interaction of the second particle with a third particle (or a set thereof), and so on. Nanocascades resemble toppling dominoes that are placed in a row. Typically, the particles in a nanocascade are in a metastable configuration that can be triggered to cascade into a lower energy configuration by other particles. By arranging the particles in suitable configurations, one can realize wires to transmit signals, or even logic gates. *Molecule cascades*, the topic of this paper, are nanocascades in which interactions take place between molecules. Researchers have succeeded [11] to create molecule cascades of Carbon Monoxide (CO) molecules arranged on a copper (Cu) surface. There is a weak repulsion between the molecules when they are in nearest grid points. This repulsion is sufficient in certain configurations to move molecules one grid point away, and it has been exploited to realize configurations with the functionality of wires, NOT-gates, and AND-gates [9]. These elements have been ingeniously combined into a 3-bit sorter circuit, of which a fascinating video is available online [12].

Though the molecule cascades in [11] 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—are not of a fundamental nature [11]. 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. 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 molecules can be arranged. Even computer-directed search through molecular configurations, however, eventually has to face the reality of combinatorial explosions, due to which many reasonably sized configurations are out of reach.

1.2 Formal methods

Fortunately, computer scientists have devoted significant efforts to develop formal methods for verifying the correct-

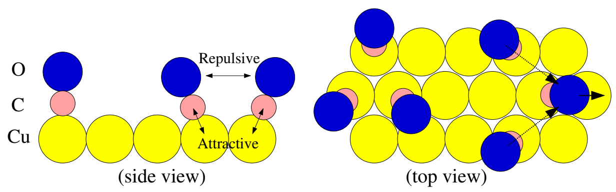


Figure 1: Tilting of CO molecules as a result of mutually repulsive forces

ness of systems faced with the state explosion problem. Symbolic methods based on Binary Decision Diagrams [4] have been crucial to verify finite-state concurrent systems in this context [15]. The problem discussed in this paper belongs to this class. In addition, model checking methods [6], which are commonly used to verify that a model (implementation) satisfies a formal specification, are used in this paper to find possible implementations, based on molecule cascades, that satisfy the behavior of logic gates. The behavior is specified using temporal logic formulae [7].

1.3 Organization of the paper

Section 2 intuitively describes the physics behind CO molecule-based interactions and computation. Section 3 introduces the formal background required to model molecular systems. Section 4 presents a generally applicable methodology used to derive molecular configurations implementing the behavior of logic gates. A case study, based on the molecule cascades in [11], is presented in Section 5. We finish with conclusions.

2. MOLECULAR SYSTEMS

The molecular systems considered in this paper consist of a surface of Copper (Cu) atoms arranged in a regular grid, such that the Cu crystal structure is cut along the 111-plane, denoted as a Cu(111) surface. 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) [16], Ni(100), Pd(100), and Cu(100) [25].

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 (Fig. 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 [16]. 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.

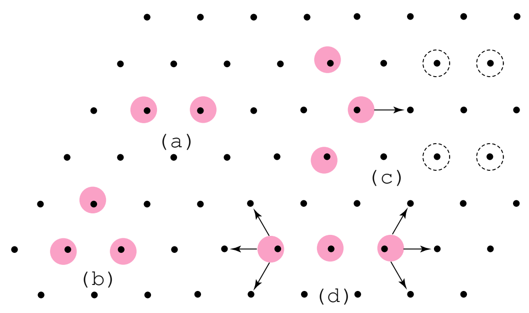


Figure 2: Stable and unstable configurations of CO molecules. Tilting of a molecule is indicated by a displacement with respect to its grid point. (a) Two CO molecules in their nearest neighbor configuration are stable, as well as (b) three molecules. (c) The *chevron* configuration is unstable and the 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.

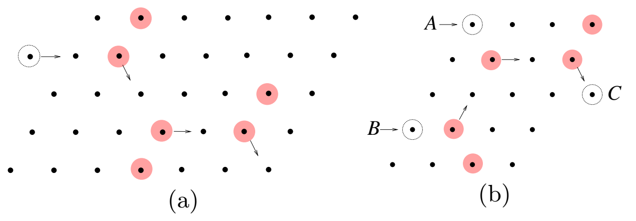


Figure 3: (a) Linked chevron cascade that can be used to propagate a signal from the upper left to the lower right, (b) Operation $C = A \wedge B$ implemented by a chevron cascade.

In many configurations of CO molecules, the carbon-copper bonding is strong enough to keep CO molecules anchored to Cu grid points, e.g., like when we have an isolated CO molecule, a pair of CO molecules at their nearest (distance 1) grid points (Fig. 2(a)), or a triangle of nearest CO molecules (Fig. 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 may tunnel to a neighboring grid point, a phenomenon that is called *hopping*. A typical example is the *chevron* configuration in Fig. 2(c), where the central molecule has tilted to the extent that it will hop one grid point to the right. Fig. 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 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 [9, 11]. Fig. 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. Fig. 3(b) shows an AND-gate based on the chevron interaction.

As pointed out in [11], 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 [11]. No fundamental arguments exist to preclude the existence of a reset mechanism inherent in a molecular system.

3. ABSTRACT MODEL OF MOLECULAR SYSTEMS

We introduce an abstract model to represent the behavior of molecular systems.

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[1]. We also use implication operators

$$\begin{aligned} a \Rightarrow b &\equiv \neg a \ \vee\ b \\ a \Leftrightarrow b &\equiv (a \wedge b) \vee (\neg a \wedge \neg b) \end{aligned}$$

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

Given two functions F and G, we say that $F \leq G$ if $F(X) = 1 \Rightarrow G(X) = 1$ for any input assignment X. Given a function $F(x_1, \dots, x_n)$, the positive and negative *cofactors* of F with respect to x_i are defined as

$$\begin{aligned} F_{x_i} &= F(x_1, \dots, x_i = 1, \dots, x_n) \\ F_{\overline{x}_i} &= F(x_1, \dots, x_i = 0, \dots, x_n) \end{aligned}$$

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

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

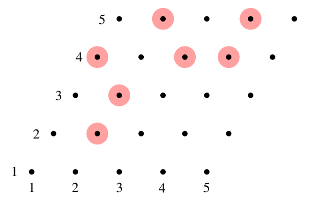
Cofactors and abstractions can be naturally extended from one variable to sets of variables.

A function is *positive unate* in variable x_i if $F_{\overline{x}_i} \leq F_{x_i}$, and it is *negative unate* in variable x_i if $F_{x_i} \leq F_{\overline{x}_i}$.

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 $s \to s'$. A *path* of length n in $\mathcal{M}$ is a sequence of transitions $s \to s_1$, $s_1 \to s_2$, ..., $s_{n-1} \to s_n$, and is represented by $s \xrightarrow{n} s_n$. A path of any length (including zero

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



$$\begin{aligned} s \ = \ & \\ & \overline{x}_{51} \cdot x_{52} \cdot \overline{x}_{53} \cdot x_{54} \cdot \overline{x}_{55} \cdot \\ & x_{41} \cdot \overline{x}_{42} \cdot x_{43} \cdot x_{44} \cdot \overline{x}_{45} \cdot \\ & \overline{x}_{31} \cdot x_{32} \cdot \overline{x}_{33} \cdot \overline{x}_{34} \cdot \overline{x}_{35} \cdot \\ & \overline{x}_{21} \cdot x_{22} \cdot \overline{x}_{23} \cdot \overline{x}_{24} \cdot \overline{x}_{25} \cdot \\ & \overline{x}_{11} \cdot \overline{x}_{12} \cdot \overline{x}_{13} \cdot \overline{x}_{14} \cdot \overline{x}_{15} \end{aligned}$$

Figure 4: State of a molecular system

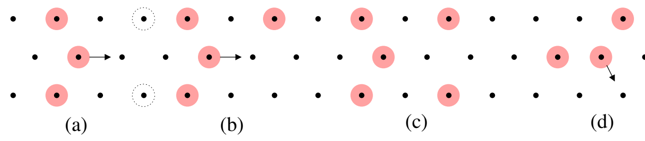


Figure 5: Possible transitions of configurations

length) is represented by $s \xrightarrow{*} s'$. A state s in $\mathcal{M}$ is said to be *reachable* if there is a path $s_0 \xrightarrow{*} s$.

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 5x5-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).

3.4 Hopping of molecules (transitions)

The molecule-hopping mechanism illustrated in Fig. 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 $s \to s'$ 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 Fig. 5(a) is unstable unless there are two molecules at the points holding the dotted circles. The configuration in Fig. 5(b) is still unstable, but the one in Fig. 5(c) is stable, since the 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 configurations in Figs. 5(a-c) can be characterized by the following Boolean functions:

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

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 Fig. 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 Fig. 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 is required [6]. This paper employs Computational Tree Logic (CTL) [7] 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 [4] 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

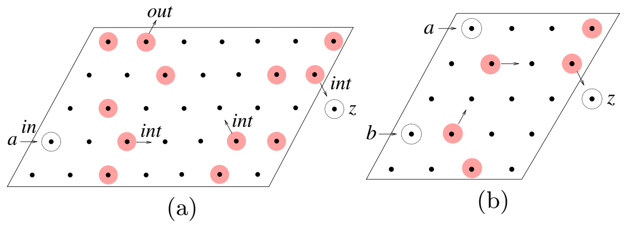


Figure 6: (a) Classification of the rules, (b) Example of AND gate

dynamics of the underlying system. $\mathcal{R}$ can be partitioned into three categories (see Fig. 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 Fig. 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 Fig. 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:

$$\textsc{Good\_Interface}(X) = \bigwedge_{r\in\mathcal{R}_{out}} AG(\neg\textsc{Enabling}_r(X))$$

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[2]. To avoid visiting such configurations, the analysis is reduced to few specific patterns whose behavior can be predicted either analytically or experimentally (e.g. using an STM, like in [11]). 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

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

to these undesired configurations. The formula $\textsc{Safe}(X)$ ensures this:

$$\textsc{Safe}(X) = \bigwedge_{c \in \mathcal{U}} AG(\neg c(X))$$

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

$$\textsc{Gate\_True}(X) = AF\ AG(z).$$

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 never holds a molecule, i.e.:

$$\textsc{Gate\_False}(X) = AG(\neg z).$$

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:

$$(a \wedge b) \Rightarrow \textsc{Gate\_True}(X)$$

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.

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 in the Appendix.

4.2 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:

- Configurations with the minimum number of molecules.
- Configurations using the minimum number of hopping rules.
- Configurations minimizing the *critical path* length (number of hops from inputs to output).
- Configurations with multiple-entry inputs, i.e. gates in which each input can enter the gate through multiple locations.

The gates with the minimum number of molecules can for example be obtained by finding the configuration with the minimum number of positive literals. This problem reduces to finding the shortest path in the BDD that characterizes all configurations [19]. Lack of space prevents us from going into details of how to include specific properties; we will, however, give some examples in Section 5.

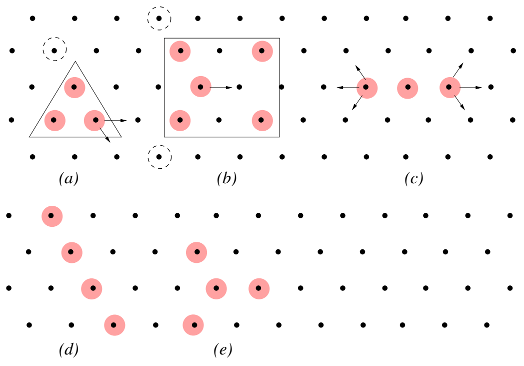


Figure 7: (a-c) Rules derived from [9], (d-e) Prohibited patterns

5. A CASE STUDY: IBM CASCADES

In this section, we derive a universal library of gates for the IBM molecular cascades. The library has been automatically obtained by the techniques of the previous section.

5.1 Rules and unsafe patterns

Apart from the main rules in Fig. 2, which are inspired by the heuristic rules in [9], there are configurations (Fig. 7), including unsafe ones, that need a careful consideration.

Fig. 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 [9]). For the sake of completeness, we add a rule corresponding to the behavior in Fig. 7(a).

The configuration in Fig. 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 Fig. 7(c) is highly nondeterministic, as we have seen in

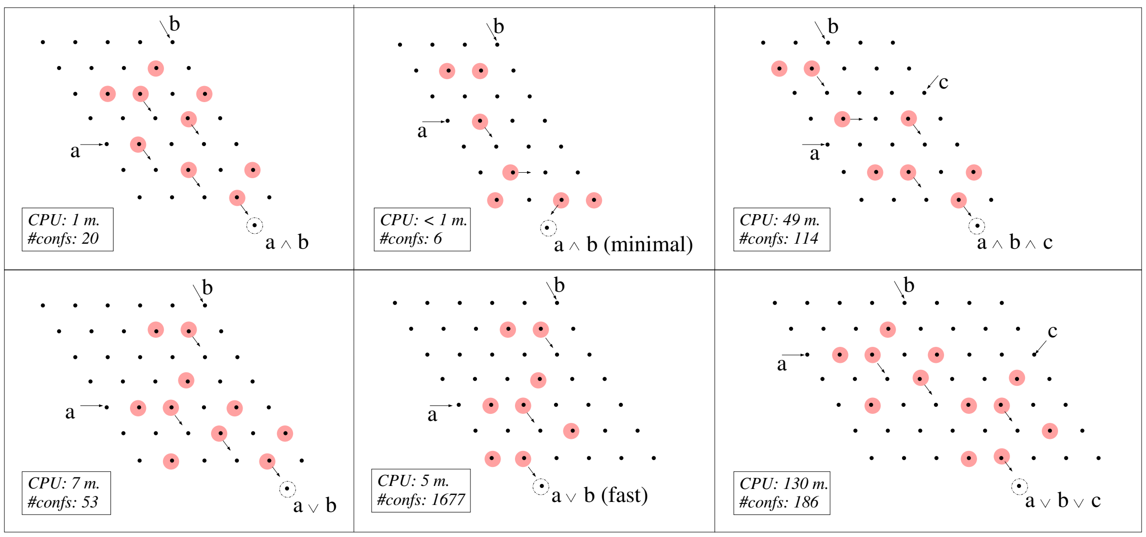


Figure 8: Library of gates: three implementations for the AND and OR gates

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

Table 1: A complete family of dual-rail gates

Section 2. Even if this rule could be considered legal, the chances of finding a gate using it are extremely small[3]. For this reason, we have declared this rule *unsafe*.

Finally, the two configurations shown in Figs. 7(d-e) describe situations with highly unpredictable behavior, given the complexity of the interactions between the molecules. Accordingly, these configurations have been declared *unsafe* as well.

5.2 Complete library of dual-rail gates

Molecule cascades can implement positive unate gates in a natural way. Negative gates (e.g. NOT, NAND, NOR) are difficult to implement, however, since this would require the output molecule to "untopple" when the input molecules topple [9]. Dealing with this problem requires that negative functions are implemented by positive gates, and delay-insensitive codes [26] are very helpful to this end. The most popular of these is *dual-rail encoding* [20], 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.

Table 1 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.

Fig. 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 to another. 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[4] 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 Fig. 8, configurations satisfying a certain property (see Section 4.2) 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 Fig. 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, and this explains the increased requirements in CPU time.

The configuration in Fig. 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 [9], and it does not work for the (1,1) assignment either. The construction of a crossover gate that works correctly for all logic combinations seems 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 [11]. Using our systematic approach, we did not find a complete crossover

[3]Actually, no gate in [9] uses this rule.

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

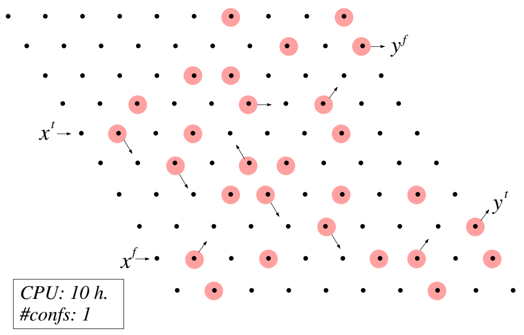


Figure 9: A dual-rail NOT gate

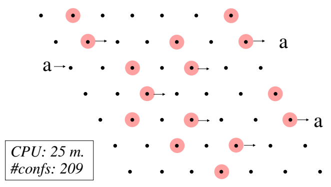


Figure 10: Fork

that works for all logic combinations either, even though we let the system run on several different grid spaces. 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.

A cell to fork a signal is presented in Figure 10. It has the same height as the gates in the library.

6. CONCLUSIONS

We have proposed a systematic method to find functional structures of molecules, given the interaction rules governing the molecules. Since the rules are represented by logic formulae, there is a limitation to the range of rules that can be handled by the 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 molecular systems may be less suitable to such an abstraction than others.

We have applied our method to find logic gates based on CO molecules on a Cu(111) surface. Though very fascinating to computer scientists, this context may prove of limited practical value—at least in the coming decade—for the realization of successors of integrated circuits. On a shorter term, however, there are other applications that may profit from our method. These applications may be characterized by the need to check out a large number of molecular configurations in order to obtain those with certain desired functionalities, whereby, once found, the speed by which transitions in these configurations take place is of less importance. Applications in surface chemistry come to mind at first, but wider applicability in chemistry may even be possible, since our method is not restricted to the regular topologies typical of metallic surfaces or to any homogeneity in the types of molecules used. The proposed method may thus play an important role in guiding material science research, and it may even suggest unexpected molecular configurations and interactions that initiate fruitful follow-up studies. In this context, CAD tools and formal models may be indispensible to deal with large and complex systems.

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. The design of molecular structures that are reusable or reversible (e.g. [2]) will require further research. 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 [10].

Acknowledgments

We would like to thank Dr. Yoshishige Okuno, Dr. Shukichi Tanaka, and Dr. Shiyoshi Yokoyama of the Nano ICT group at the National Institute of Information and Communications Technology (NICT), Japan, for their advice and discussions about molecular systems. This work has been supported by CICYT TIN2004-07925.

7. REFERENCES

[1] P. Avouris, J. Appenzeller, R. Martel, and S. Wind. Carbon nanotube electronics. *Proc. of the IEEE*, 91(11):1772–1784, 2003.

[2] C. Bennett. The thermodynamics of computation—a review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.

[3] G. Bourianoff. The future of nanocomputing. *Computer*, 36(8):44–53, August 2003.

[4] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[5] J. Burch, E. Clarke, D. Long, K. McMillan, and D. Dill. Symbolic model checking for sequential circuit verification. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, 1994.

[6] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[7] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[8] L. Durbeck and N. Macias. The cell matrix: an architecture for nanocomputing. *Nanotechnology*, 12(3):217–230, September 2001.

[9] D. M. Eigler, C. P. Lutz, M. F. Crommie, H. C. Mahoran, A. J. Heinrich, and J. A. Gupta. Information transport and computation in nanometre-scale structures. *Phil. Trans. R. Soc. Lond. A*, 362(1819):1135–1147, 2004.

[10] M. Frank. Introduction to reversible computing: motivation, progress, and challenges. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 385–390, New York, NY, USA, 2005. ACM Press.

[11] A. J. Heinrich, C. P. Lutz, J. A. Gupta, and D. M. Eigler. Molecule cascades. *Science*, 298:1381–1387, 2002.

[12] IBM scientists build world's smallest operating computing circuits. http://domino.watson.ibm.com/comm/pr.nsf/pages/

news.20021024_cascade.html, Oct. 2002.

[13] *International Technology Roadmap for Semiconductors*, 2005.

[14] C. Joachim, J. Gimzewski, and A. Aviram. Electronics using hybrid-molecular and mono-molecular devices. *Nature*, 408:541–548, 30 November 2000.

[15] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.

[16] H. Kato, H. Okuyama, S. Ichihara, and M. Kawai. Lateral interactions of CO in the (2×1)p2mg structure on Pd(110): Force constants between tilted CO molecules. *Journal of Chemical Physics*, 112(4):1925–1936, 22 January 2000.

[17] C. Lent, B. Isaksen, and M. Lieberman. Molecular quantum-dot cellular automata. *J. Am. Chem. Soc.*, 125(4):1056–1063, 2003.

[18] K. K. Likharev and D. B. Strukov. Cmol: Devices, circuits, and architectures. In G. C. et al. (eds.), editor, *Introduction to Molecular Electronics*, pages 447–477. Springer, Berlin, 2005.

[19] B. Lin and F. Somenzi. Minimization of symbolic relations. In *ICCAD*, pages 88–91, Nov. 1990.

[20] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, 1959.

[21] Y. Ono, A. Fujiwara, K. Nishiguchi, H. Inokawa, and Y. Takahashi. Manipulation and detection of single electrons for future information processing. *Journal of Applied Physics*, 97:031101–1–031101–19, 2005.

[22] F. Peper, J. Lee, S. Adachi, and S. Mashiko. Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers? *Nanotechnology*, 14(4):469–485, April 2003.

[23] S. D. Sarma, J. Fabian, X. Hu, and I. Zutic. Issues, concepts, and challenges in spintronics. In *Proc. 58th IEEE Device Research Conf. (DRC)*, pages 95–98. IEEE, June 2000.

[24] M. Stone. The representation theorem for Boolean algebras. *Trans. Amer. Math. Soc.*, 40:37–111, 1936.

[25] P. Uvdal, P.-A. Karlsson, C. Nyberg, and S. Andersson. On the structure of dense CO overlayers. *Surface Science*, 202(1-2):167–182, 1 August 1988.

[26] T. Verhoeff. Delay-insensitive codes—an overview. *Distributed Computing*, 3(1):1–8, 1988.

APPENDIX

A. SYMBOLIC ANALYSIS OF SYSTEMS

A.1 Binary Decision Diagrams

Binary Decision Diagrams (BDDs) [4] 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 usually 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 [24] 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.

A.2 Symbolic reachability analysis

This section describes the classical symbolic algorithm to calculate the reachable states of a finite-state system [15].

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 (X and Y are disjoint sets of variables). 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

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

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

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

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.

A.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, i.e.:

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

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* [5]. 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 the small transition relations of each rule. This strategy improves the efficiency of the traversal algorithm significantly.