**Quorum Sensing and Verification in Chemical Reaction Networks**

James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Hugh D. Potter, and Matthew R. Riley

Iowa State University, Ames, IA 50011, USA
jil,lutz,rlutz,hdpotter,mrriley@iastate.edu

**Abstract** Some species of bacteria exhibit quorum sensing, a phenomenon in which gene expression and resultant behavior undergo a phase change at some threshold population density. We show here that even very simple nanodevices, modeled as chemical reaction networks, can by a different mechanism also sense their own population densities and change their behaviors dramatically at certain thresholds. Moreover, the magnitudes of these thresholds can thwart attempts to use model checking, simulation, or approximation by differential equations to formally verify the behaviors of such nanodevices at realistic population densities. We show how formal theorem provers can successfully verify some such systems at sizes where other verification methods fail.

**Keywords:** Chemical reaction network · Formal verification · Molecular programming · Quorum sensing

1 **Introduction**

Molecular programming is a growing field that creates useful nanoscale systems via the information-processing capabilities of DNA and other biomolecules. Molecular programming applications include diagnostic biosensors, medical therapeutics and molecular robots. Chemical reaction networks (CRNs) are a mathematical abstraction similar to Petri nets, used as a programming language to specify the dynamic behavior of these systems. Existing software can compile these chemical reaction networks into DNA strand displacement systems that simulate them [18,6,2,19].

In this paper we show that even very simple nanodevices, modeled as chemical reaction networks, can exhibit quorum sensing-like behavior that thwarts some approaches to formally verifying their behaviors. Specifically, we present as a running example a stochastic chemical reaction network $N$ with the following properties:

1. Every reaction of $N$ has two reactants (input molecules) and two products (output molecules). That is, $N$ is a population protocol [1].
2. \( N \) consists of a simple nanodevice and is initialized to have some positive integer \( N \) of copies of this device.

3. For all \( N < 10^{10} \) and all \( N > 10^{20} \), essentially the entire population of \( N \) will eventually consist of molecules of a single \textit{red} species.

4. For all \( 10^{10} < N < 10^{20} \), essentially the entire population of \( N \) will eventually consist of molecules of a single \textit{blue} species.

The thresholds \( 10^{10} \) and \( 10^{20} \) here imply (see Figure 1) that straightforward attempts to model check \( N \) or simulate \( N \) will conclude that \( N \) converges to \textit{red}, and approximations of \( N \) by differential equations will arrive at the same conclusion. But many if not most realistic molecular programming experiments and applications will have device counts strictly between \( 10^{10} \) and \( 10^{20} \), and \( N \) converges to \textit{blue} in this range, contrary to what these three verification methods would indicate. This limitation on the use of these verification methods in molecular programming does not arise in more traditional software or hardware applications.

Fortunately, theorem proving can often deliver correct verdicts even when population sizes defeat the above three methods. We show here how automated theorem proving can verify the region of realistic nano-experiments (property 4 above). The techniques used in Isabelle to prove that this region is correct can also be used to prove the red regions (property 3 above) are also correct.

Quorum sensing systems detect and respond to the population density of certain molecules \cite{133}. The behavior of a quorum sensing system differs depending on whether the population density is above or below a certain threshold. Many molecular programmed systems, such as those for in vivo health monitoring, will depend on the correct behavior of quorum sensing.

Especially because many planned applications of molecular programming are safety-critical, such as in vivo health monitoring systems and targeted drug delivery systems, verification of behavioral properties is necessary. Experimental verification in the laboratory or with human subjects is costly, time-consuming, and intrinsically partial. As with the testing of software and hardware, edge and corner case behaviors may not reveal themselves in the necessarily limited number of executions. Moreover, we would like to find errors earlier than the
implementation of the system in molecules. Discovering bugs prior to implement-
ation simplifies fixing the fixes and reduces the cost and effort of system
development. Verification that the CRN design of the molecular programmed
system matches its intended behavior thus becomes essential.

Many molecular programmed systems have populations in excess of \(10^{10}\)
devices. This scale poses a serious challenge for verification of molecular pro-
grammed system designs. Current formal verification methods cannot handle
such scales. To verify molecular programmed systems, researchers may model
check a small population count (typically no more than a few hundred) and/or
simulate the behavior of a larger but still not realistic population count (up to
a few million). Another approach is to approximate the behavior of exceedingly
large population counts with ordinary differential equations (ODEs).

Trying to verify a system whose behavior changes at a high threshold pop-
ulation density by considering a few devices (via model checking) or by testing
its behavior with a medium-sized count of devices (via simulation) or by using
ODEs to check an unrealistically large number of devices leaves a gap in the
middle, as Figure 1 shows. That is, current approaches do not cover the range of
possible behaviors for actual systems. In fact, some systems display one behav-
ior at smaller sizes and a very different behavior at sizes that cannot currently
be checked. This is problematic because assurance of the behavior requires con-
sideration at the real-world scale. In Sect. 2 we describe one such system, a
molecular quorum sensing system that has one behavior at small and exceed-
ingly large population counts and a very different behavior at population counts
in between, such as those of actual systems.

Model checking, simulation, or ODEs may be unable to verify the behavior of
the number of molecular devices contained in a real-world system. The question
thus arises of how to verify the behavior of a system where the population (the
number of devices) matters. This paper shows that automated theorem proving
of a molecular system’s CRN design model can address this goal. The paper
demonstrates the application of our proposed technique on our CRN design
model of a molecular quorum-sensing system. We show how automated theorem
provers can successfully verify some such molecular systems at sizes where other
methods fail.

2 The Chemical Reaction Network \(N\)

Chemical reaction networks (CRNs) are abstract models of molecular processes
in well-mixed solutions. They are roughly equivalent to three models used in
distributed computing, namely, Petri nets, population protocols, and vector ad-
dition systems [7]. This paper uses stochastic chemical reaction networks.

For our purposes, a (stochastic) chemical reaction network \(N\) consists of
finitely many reactions, each of which has the form

\[
A + B \rightarrow C + D,
\]

(2.1)

where \(A, B, C,\) and \(D\) (not necessarily distinct) are species, i.e., abstract types
of molecules. Intuitively, if this reaction occurs in a solution at some time, then
one $A$ and one $B$ disappear from the solution and are replaced by one $C$ and one $D$, these things happening instantaneously. A state of the CRN $N$ at a particular moment of time consists of the nonnegative integer counts $a, b, \ldots$ of the species $A, B, \ldots$ of $N$ at that moment. Note that we are using the so called “lower-case convention” for denoting species counts.

In the full stochastic CRN model, each reaction also has a positive real \textit{rate constant}, and the random behavior of $N$ obeys a continuous-time Markov chain derived from these rate constants. However, our results here are so robust that they hold for \textit{any} assignment of rate constants to our running example, so we need not concern ourselves with rate constants or continuous-time Markov chains. In fact, for this paper, we can consider the reaction (2.1) to be the if-statement

$$\text{if } a > 0 \text{ and } b > 0 \text{ then } a, b, c, d := a - 1, b - 1, c + 1, d + 1,$$ (2.2)

where “:=” is parallel assignment. An execution of $N$ is then any sequence of such reactions that is \textit{fair} in the sense that the if-condition of a reaction cannot remain true forever without the reaction being executed.

The fact that each reaction (2.1) has two \textit{reactants} ($A$ and $B$) and two \textit{products} ($C$ and $D$) means that $N$ is a \textit{population protocol} \cite{1}. In particular, this condition implies that the total population of all species never changes in the course of an execution.

Intuitively, if we initialize a CRN by assigning initial counts to each of its species, and if this initialization gives the species $A, B, \ldots$ initial counts $a_0 \cdot N, b_0 \cdot N, \ldots$, where the coefficients $a_0, b_0, \ldots$ have greatest common divisor 1, then we can regard the CRN with this initialization as a solution consisting of $N$ copies of the individual nanodevice corresponding to the case $N = 1$.

Our running example, which we now define, exploits the fact that the nonlinearity of CRNs implies that the behavior $N$ nanodevices may be very nonlinear in $N$. The CRN is defined as follows:

$$Z_i + Z_i \rightarrow Z_{i+1} + R, \ i \in [0, 33] \quad (2.3)$$

$$Z_i + Z_i \rightarrow Z_{i+1} + B, \ i \in [34, 64] \quad (2.4)$$

$$Z_i + R \rightarrow Z_i + B, \ i \in [34, 65] \quad (2.5)$$

$$Z_{65} + Z_{65} \rightarrow Q + R \quad (2.6)$$

$$Q + B \rightarrow Q + R \quad (2.7)$$

$$Q + Z_i \rightarrow Q + Q, \ i \in [0, 65] \quad (2.8)$$

$$Q + Q \rightarrow Q + R \quad (2.9)$$

This CRN is initialized with $z_0 = N$ and $z_i = q = r = b = 0, i \in [1, 65]$, and begins in the lower phase. If $N < 2^{34}$ the CRN will remain in the lower phase until it terminates. In this case the CRN will create red molecules via reactions
and terminate with \( r \geq N - \log(N) \). When \( N \geq 2^{34} \approx 10^{10} \) the CRN will eventually transition into the middle phase by creating a copy of \( Z_{34} \). Once this happens the CRN will begin to execute reactions (2.5) that catalytically convert red molecules to blue. If \( N \geq 2^{66} \approx 10^{20} \) the CRN will ultimately transition into the upper phase by creating a copy of \( Q \). Once at least one copy of \( Q \) is present the CRN will begin to catalytically convert all molecules to red (2.7 - 2.9).

### 3 Model Checking

For model checking, the probabilistic model checker PRISM provides verification of properties on a system composed of several devices [10]. Kwiatkowska and Thachuk used PRISM for the probabilistic verification of CRNs for biological systems using the probabilistic model checker PRISM [11]. Recent work by Cauchi, et al. using formal synthesis allowed verification of systems with 10 continuous variables [5].

Using PRISM we verified that the Prism property “\( P > 0 \implies [F \ G \ r > 190] \)” held for the CRN design model for an initial population of \( z_0 = 200 \). However, even with 8 GB of memory, we were unable to model check it for an initial population of \( z_0 = 400 \). Depending on the CRN, for population counts over 5 or so, state-space explosion may be a problem. Methods suggested to prune the model so that meaningful model checking can occur include symmetry reduction [8], statistical model checking [4], and automated partial exploration of the model [16].

### 4 Simulation

For simulation, MATLAB’s SimBiology package is widely used to explore the behavior of a larger number of devices executing concurrently [12]. To simulate the CRN design model, we used Visual GEC. Visual GEC is a software tool for simulation and analysis based on the GEC language [17], and it can export models to PRISM. Simulation with Visual GEC with an initial population of \( z_0 = 64,000,000 \) molecules was not able to verify the Blue behavior of the CRN correctly. Simulation with Visual GEC also caused a memory allocation error when \( z_0 = 128,000,000 \). Matlab Simbiology also was not able to simulate the system, generating an out of memory exception.

### 5 Differential Equations

We have seen how model checking and simulation fail to detect behaviors in our quorum sensing CRN due to the processing time required for a large number of input molecules. We now demonstrate that our quorum sensing CRN example also fails to detect behaviors when approximated by deterministic mass action semantics. In this case it is not due to computation power. In this model, reactions are modeled using differential equations, and the set of reactions in a
6 J. Lathrop et al.

CRN is modeled by a system of polynomial autonomous differential equations. A mathematical connection between the stochastic model and the deterministic model is given by Kurtz [9], where he shows that the stochastic model is equivalent to the deterministic model as the number of molecules in the system grows towards infinity.

In general, the system of differential equations induced by a CRN is difficult or impossible to solve exactly, and numerical methods are often used to approximate solutions. Here, we utilize Matlab and the SimBiology package [12] to numerically integrate the resulting system of differential equations. Researchers model CRNs using deterministic mass action semantics under the supposition that there are infinitely many molecules, relying on the idea that the CRN behavior is similar to low molecule counts in the stochastic model. This is demonstrated below in Figure 2 where over time the Matlab numerical solution of the system of differential equations yields a large concentration of red species, missing the fact that the stochastic behavior at intermediate molecular counts acts in an opposite manner.

Figure 2. Numerical integration of the CRN using Matlab and Simbiology package.

6 Theorem Proving

We used the interactive theorem prover Isabelle [15,14] to validate the use of theorem provers with CRNs at scale. For the quorum sensing CRN here, we
theory termin
  imports qsvcrn
begin

definition terminal :: state ⇒ bool where
terminal s1 = (¬ (∃ s2. K s1 s2))
definition nonterm :: state ⇒ bool where
nonterm s = (¬ (terminal s))
definition path-term :: (nat ⇒ state) ⇒ bool where
(path-term p) = (∃ t. (terminal (p t)))
definition state-term :: (state ⇒ bool) where
(state-term s) = (∀ p :: (nat ⇒ state).
(∃ t. ((p t) = s)) → (path-term p)))

lemma dec-imp-term:
  fixes f :: state ⇒ nat
  fixes p :: nat ⇒ state
  fixes c :: nat
  assumes evterm: ((f s) ≤ c) → (terminal s)
  assumes dec: ∀ i. ((¬ (terminal (p i))) → (f (p (i + 1))) < (f (p i))))
  shows path-term p
proof −
  { fix n::nat
    have ((∃ t. ((f (p t)) ≤ n)) → (path-term p))
    proof (induction n)
      case 0
      then show ?case
      using dec gr-implies-not0 path-term-def by blast
    next
      case Suc n
      then show ?case
      using (metis dec le-SucE less-Suc-eq-le path-term-def) by blast
    qed
  } then show ?thesis by blast
qed
end

Figure 3. A sample Isabelle lemma that helps prove that the CRN terminates.
verified via the Isabelle theorem prover that if the initial population in $z_0$ is in the blue region (between $2^{34}$ and $2^{66} - 1$) Isabelle correctly proves that the system is flooded with blue ($b$) molecules. Figure 3 shows a sample of the Isabelle session code that helps prove that the CRN terminates (no reaction can occur) for input sizes in the blue region. This demonstrates that using a theorem prover to verify CRN operation is a viable and useful technique to verify correctness in CRNs, especially at those scales where other verification techniques fail.

**Acknowledgment**

We thank Neil Lutz for technical assistance.

**References**

1. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing 20(4), 279–304 (2007)
2. Badelt, S., Shin, S.W., Johnson, R.F., Dong, Q., Thachuk, C., Winfree, E.: A general-purpose CRN-to-DSD compiler with formal verification, optimization, and simulation capabilities. In: Proceedings of the 23rd International Conference on DNA Computing and Molecular Programming. pp. 232–248. Lecture Notes in Computer Science (2017)
3. Bassler, B.L.: How bacteria talk to each other: regulation of gene expression by quorum sensing. Current Opinion in Microbiology 2(6), 582 – 587 (1999). [https://doi.org/10.1016/S1369-5274(99)00025-9](http://www.sciencedirect.com/science/article/pii/S1369527499000259)
4. Cardelli, L., Kwiatkowska, M., Whitby, M.: Chemical reaction network designs for asynchronous logic circuits. Natural Computing 17(1), 109–130 (2018). [https://doi.org/10.1007/s11047-017-9665-7](10.1007/s11047-017-9665-7)
5. Cauchi, N., Laurenti, L., Lahijanian, M., Abate, A., Kwiatkowska, M., Cardelli, L.: Efficiency through uncertainty: scalable formal synthesis for stochastic hybrid systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019. pp. 240–251 (2019). [https://doi.org/10.1145/3302504.3311805](10.1145/3302504.3311805)
6. Chen, Y.J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., Seelig, G.: Programmable chemical controllers made from DNA. Nature Nanotechnology 8(10), 755–762 (2013)
7. Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of chemical reaction networks. In: Condon, A., Harel, D., Kok, J.N., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses, pp. 543–584. Natural Computing Series, Springer (2009)
8. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchshyn, O.: Probabilistic model checking of complex biological pathways. In: Priami, C. (ed.) Computational Methods in Systems Biology. pp. 32–47. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
9. Kurtz, T.G.: The relationship between stochastic and deterministic models for chemical reactions. The Journal of Chemical Physics 57(7), 2976–2978 (1972)
10. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proceedings of the 23rd International Conference on Computer Aided Verification. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (2011)
11. Kwiatkowska, M., Thachuk, C.: Probabilistic model checking for biology. Software Systems Safety 36, 165–189 (2014)
12. MATLAB: version 9.5.0 (R2018b). The MathWorks Inc., Natick, Massachusetts (2018)
13. Miller, M.B., Bassler, B.L.: Quorum sensing in bacteria. Annual Review of Microbiology 55(1), 165–199 (2001). https://doi.org/10.1146/annurev.micro.55.1.165, pMID: 11544353
14. Nipkow, T., Klein, G.: Concrete Semantics. Springer International Publishing, 1 edn. (2014)
15. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL, Lecture Notes in Computer Science, vol. 2283. Springer-Verlag Berlin Heidelberg, 1 edn. (2002)
16. Pavese, E., Braberman, V., Uchitel, S.: Less is more: Estimating probabilistic rewards over partial system explorations. ACM Transactions on Software Engineering and Methodology 25(2), 16:1–16:47 (2016)
17. Pedersen, M., Phillips, A.: Towards programming languages for genetic engineering of living cells. Journal of The Royal Society Interface (2009)
18. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proceedings of the National Academy of Sciences 107(12), 5393–5398 (2010)
19. Thubagere, A.J., Thachuk, C., Berleant, J., Johnson, R.F., Ardelean, D.A., Cherry, K.M., Qian, L.: Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. Nature Communications 8 (2017)