Challenges in automated verification and synthesis for molecular programming

Marta Kwiatkowska
Department of Computer Science, University of Oxford, UK
marta.kwiatkowska@cs.ox.ac.uk

Abstract

Molecular programming is concerned with building synthetic nanoscale devices from molecules, which can be programmed to autonomously perform a specific task. Several artifacts have been demonstrated experimentally, including DNA circuits that can compute a logic formula and molecular robots that can transport cargo. In view of their natural interface to biological components, many potential applications are envisaged, e.g. point-of-care diagnostics and targeted delivery of drugs. However, the inherent complexity of the resulting biochemical systems makes the manual process of designing such devices error-prone, requiring automated design support methodologies, analogous to design automation tools for digital systems. This paper gives an overview of the role that probabilistic modelling and verification techniques can play in designing, analysing, debugging and synthesising programmable molecular devices, and outlines the challenges in achieving automated verification and synthesis software technologies in this setting.

1 Introduction

Recently, significant advances have been made in the experimental design and engineering of synthetic, biomolecular systems, such as those built from DNA, RNA or enzymes. The interest in such devices stems from the fact that they are *autonomous* – they can interact with the biochemical environment, process information, make decisions and act on them – and *programmable*, that is, they can be systematically configured to perform specific computational or mechanical tasks. The computational power of such systems has been shown to be equivalent to Turing computability (Soloveichik et al. 2010), albeit the computation itself proceeds through molecules acting as inputs, interacting with each other and producing product molecules. Experimental advances are fast accelerating, with examples that have been demonstrated including diagnostic biosensors (Jung and Ellington 2014), logic circuits built from DNA (Seelig et al. 2006; Qian and Winfree 2011), DNA-only controllers (Chen et al. 2013) and molecular robots that can deliver cargo (Yurke et al. 2000; Yin et al. 2004). Since such systems can perform information processing within living cells, their use is envisaged in healthcare applications, where safety is paramount. The fast-growing field of *molecular programming* is concerned with developing techniques to design, analyse and realise the computational potential of such programmable molecular devices. In conjunction with the DNA self-assembly capabilities, which has enabled a wide range of structure-forming technologies at the nanoscale (Rothemund 2006), the future potential of these developments is tremendous, particularly for smart therapeutics, point-of-care diagnostics and synthetic biology.

Device design is supported by electronic design automation (EDA) environments, which provides methodologies and tools to *automate* the design, verification, testing and synthesis of electronic systems from a high-level description. The software level, at which design is applied, is separate from the hardware level, e.g. fabrication, and can involve multiple levels of abstractions. In the semiconductor industry, design automation has established itself as a key technology to tackle the complexity of the designs, improve design quality, and increase reuse. In the 1990s, VLSI design was revolutionised by *formal verification*, and in particular *automated* methods such as model checking, now a key component of modern EDA tools, which ensure device safety and reliability, and significantly reduce development costs.

Molecular programming aims to devise programming languages, techniques and software tools to achieve automatic compilation of a molecular system down to the set of components that can be implemented physically at the molecular level and executed. This is analogous to the motivation for hardware description languages, e.g. VHDL, which are refined automatically, through a series of intermediate abstractions, down to a detailed physical implementation in silicon. This paper puts forward the view that formal verification will play a similar role in design automation for molecular programming. However, the latter brings with it unique challenges: the necessity to consider inherent *stochasticity* of the underlying molecular interactions, the need to state requirements in *quantitative* form, and the importance to consider *control* of molecular systems, and not just programming in the conventional sense. Therefore, *probabilistic modelling* and *automated, quantitative verification* techniques (Kwiatkowska 2007; Kwiatkowska et al. 2007, 2011), such as those already developed for systems biology (Regev et al. 2001; Heath et al. 2008; Ciocchetta and Hillston 2009; Kwiatkowska et al. 2010) in addition to tools tailored to DNA computing (Phillips and Cardelli 2009;

Aubert et al. 2014), will form a key component of design automation for molecular programming.

The paper begins by giving a brief introduction to molecular programming, illustrated by a simple example of DNA biosensing, and then reviews the current status of formal modelling and verification technologies for molecular programming, outlining the research challenges. More detail about application of automated, quantitative verification in DNA computing can be found in the tutorial paper (Kwiatkowska and Thachuk 2014) and (Lakin et al. 2012; Dannenberg et al. 2013b,a, 2014).

2 Molecular programming

The term *molecular programming* (Hagiya 2000; Winfree 2008) refers to the application of computational concepts and design methods to the field of nanotechnology, and specifically biochemical reaction systems. The idea is to design biochemical networks that can *process information* and are *programmable*, that is, can be configured to perform a given task, be it computation of a logic formula or transporting a cargo to a specified target. Chemical reaction networks (CRNs) provide a canonical notation for describing biochemical systems, based on well understood stochastic or kinetic laws, and the computational and nanorobotic mechanisms that they can implement. A *molecular program* is thus a series of reactions, for example $X+Y \to Z$, meaning that inputs (specially designed DNA strands) X and Y are transformed to produce strand Z. An example is the Approximate Majority program (Angluin et al. 2008), where, starting with given initial numbers of molecules X and Y placed in solution, with high probability the network will converge to a state that only contains the molecules that were initially in majority.

In order to implement molecular programs, DNA technologies have been developed, of which *DNA strand displacement (DSD)* (Zhang et al. 2007; Zhang and Seelig 2011) is particularly popular, since it uses only DNA molecules, is enzyme-free, and easy to synthesize chemically. Any CRN can be implemented using the limited set of DSD reactions (Soloveichik et al. 2010); in fact, the DSD realisation of Approximate Majority was experimentally demonstrated in (Chen et al. 2013) and related to the cell cycle in (Cardelli and Csikász-Nagy 2012). DSD can be used to implement logic gates, where inputs and outputs are (single) DNA strands. An example is the transducer gate designed by Cardelli (Cardelli 2010) and diagnostic biosensors of (Jung and Ellington 2014).

The promise of DNA systems is that they can interact with biological components in their local environment, including within living cells. An important application of such systems is therefore *biosensing*, a decision process that aims to

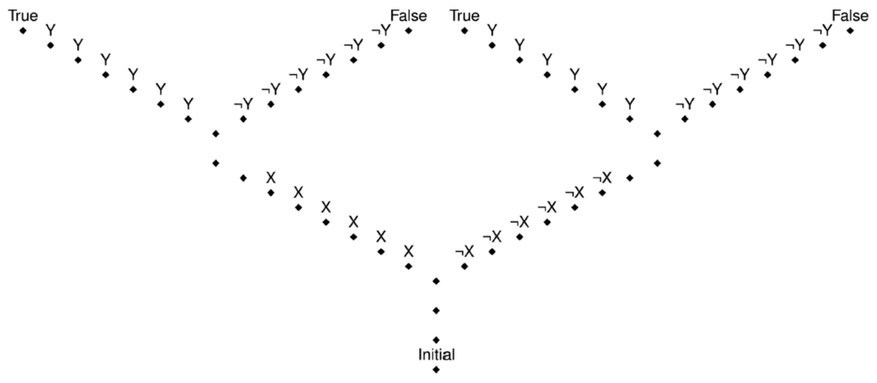

Figure 1: A DNA walker track that acts as a biosensor, shown here with six blockades.

detect various input biomarkers in an environment, such as strands of messenger RNA within a cell, and take action based on the detected input. We illustrate molecular programming applications with an example of a biosensor based on *DNA walker circuits*, which are realised using DNA strand displacement technology. DNA walkers (Wickham et al. 2011) can traverse tracks of DNA strands (*anchorages*) that are tethered to a surface, typically DNA origami tile (Rothemund 2006), taking directions at junctions that fork into two tracks, respectively labelled with X and $\neg X$. When the system is prepared, a self-consistent set of unblocking strands is added to unblock X or $\neg X$ but not both, ensuring that the walker is directed towards the target. Alternatively, the walker *senses* the strands that guide it towards the target, indicating detection and readiness to take action. The tracks can also join, and in general this type of DNA walker can be represented as a planar circuit that can compute an arbitrary Boolean function (Dannenberg et al. 2013b, 2014).

Example 1. *We consider a biosensor that detects the presence of DNA strands X and Y, and delivers cargo to the end node. Figure 1 shows the walker circuit, where the tracks labelled X and $\neg Y$ are unblocked, meaning Y is absent. At the junction, the walker selects or senses the unblocked track.*

The stepping process is illustrated in Figure 2. The walker strand carries a load (Q) that will quench fluorophores (F) that are attached to absorbing anchorages (1). It starts on the initial anchorage and, when a nicking enzyme (E) is present (2), traverses the circuit until it reaches an absorbing anchorage, which prevents further stepping. When the nicking enzyme is added to the solution, it binds to the walker-anchorage complex and cuts the anchorage into two strands. The strand formed from the tip is too short to remain hybridized to the walker, and melts away into solution. This exposes the top six nucleotides of the walker,

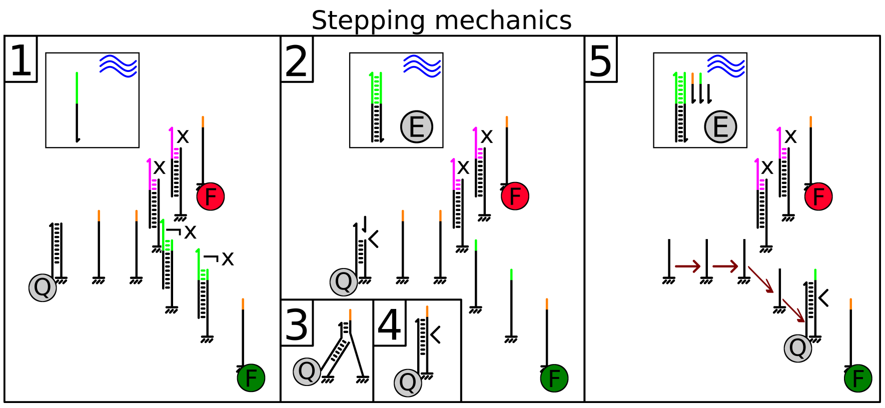

Figure 2: DNA walker circuit that can sense incoming strands (the domain coloured green).

which then attach to the next anchorage (3). In a displacement reaction, the walker becomes completely attached to the new anchorage (4). The reaction is energetically favourable, as the walker re-forms the six base-pairs with the intact anchorage that were lost after the nicking.

Repeating this process, the walker arrives at the junction. The walker continues down the track that was unblocked due to the presence of the strand being sensed (5), eventually quenching the fluorophore (F) on reaching the final node. The change in fluorescence is easily detectable, indicating that the presence of a certain configuration of molecules has been detected.

The biosensing example illustrates well the functionality that design automation for molecular programming must support. Firstly, we need *modelling frameworks* that can provide rigorous foundations to the mechanisms described by populations of molecules interacting through biochemical reactions, both in well-mixed solution, as well as localised, e.g., tethered to a surface. These must be able to capture molecular motion and structure-forming, in addition to information processing and decision making. Secondly, we need *programming languages* tailored to molecular programming and *programming abstractions*, to provide the layers through which molecular programs are transformed into the actual physical realisations. Thirdly, in view of the stochasticity, *quantitative specifications* are needed, which the designed molecular programs must meet, such as ensuring that “the probability of incorrect detection is tolerably low”, and “the expected time for the walker to reach the target node is sufficiently fast”. Finally, a broad range of *analysis* and *synthesis* methods is necessary, where the former should focus on predictability, correctness and resource usage, and the latter will need to cover automated synthesis of both the mechanism as well as the layout, as in DNA walker circuits.

3 Design automation for molecular programming

Modelling frameworks. A computational process is typically formalised by defining a transition system comprising a set of system states and a formal set of rules that govern the evolution of the system over time. A molecular program combines *discrete, continuous and stochastic* dynamics. The class of models that naturally captures such behaviours is known as stochastic hybrid systems. In view of their complexity and intractability, the modelling frameworks for stochastic hybrid systems resort to discretisation or approximation, which, under suitably strong conditions, can reduce the system model to, e.g., a finite-state Markov chain variant or a coupled system of linear equations, which are all tractable. When in solution, molecular networks, such as DNA strand displacement networks above, induce discrete stochastic dynamics; if spatiality is included, however, the continuous dimension must also be integrated to model motion in the physical space. Another important requirement is the need to augment models with quantitative features specific to molecular programming, e.g., thermodynamics, kinetics and resource usage.

There are two established frameworks for modelling molecular reactions in solution, the *continuous deterministic* approach and the *discrete stochastic* approach (Kurtz 1972; Gillespie 1977; McAdams and Arkin 1997). In the deterministic approach, one approximates the number of molecules by a continuous function that represents the time-dependence of the molecular concentration and evolves according to differential equations (ODEs) based on mass action kinetics. The ODE approach is suitable for modelling averaged behaviour and assumes large numbers of molecules. The discrete stochastic framework, on the other hand, models the stochastic evolution of populations of molecules. Reactions are discrete, stochastic events governed by rates that are typically assumed to be dependent only on the number of molecules: the system is conveniently represented as a continuous-time Markov chain (CTMC). This approach is more accurate in cases where the number of molecules is small and the system behaviour becomes non-continuous. It is also appropriate when it is necessary to take account of abstract spatial information, such as track layout in the case of molecular walkers.

The modelling challenge: State-based abstractions of molecular programs have the potential to enable analysis of correctness of the computation performed by the molecular program, for example deadlock, presence or absence of a given strand, or termination. The key challenge is *scalability*, which can be improved by identifying suitable model reductions, for example based on bisimulation quotients, symmetry reductions or symbolic techniques, or compositional theories. Another difficulty is the need to *integrate* the discrete, continuous and stochastic dynamics within a tractable modelling framework, in order to model molecular robotic systems and origami folding.

Languages for molecular programming. A number of programming languages exist that are specifically tailored to DNA computation, for example Visual DSD (Phillips and Cardelli 2009), from which CTMC models are automatically generated; Caltech's Seesaw Compiler, which accepts descriptions of logic circuits and outputs DNA sequences; and DACCAD (DNA Artificial Circuits Computer-Assisted Design) (Aubert et al. 2014), which outputs reaction networks in ODE semantics. The tools also output SBML format for further processing. One advantage of textual design descriptions is their flexibility and ease of modification, with the view to enable design reusability. Another advantage is that, as for conventional circuit designs, a range of analysis techniques are available to check for correctness of the designs, in addition to specialised properties of DNA systems such as sequencing thermodynamic properties, e.g. NUPACK, and structural descriptions for origami designs, as supported by CadNano. More generally, a variety of stochastic process algebras supported by software tools, such as stochastic pi-calculus (Regev et al. 2001) and BioPEPA (Ciocchetta and Hillston 2009), are capable of modelling molecular networks. Systems biology tools such as COPASI (Hoops et al. 2006) are also often applicable.

The language challenge: Though existing process-algebraic languages have proved useful for describing complex molecular programs, much more effort is needed to design high-level languages for emerging *experimental phenomena*, such as localised hybridization, origami design and molecular motors, to capture aspects such as spatiality, geometry and mobility, together with associated rigorous semantics, equivalence/refinement notions, and compositional behavioural theories.

Programming abstractions for design automation. Design automation tools implement design flows that compile high-level system descriptions, via intermediate languages, down to the physical design. Typically, at the top level, designs will be given in the form of a Boolean function or component-based designs, and compiled into *intermediate notations*, such as the CRN level and the sequence level, before they can be physically realised through nanofabrication. An example intermediate language is Cardelli's Strand Algebra (Cardelli 2010), a stochastic process algebra supported by the DSD tool (Phillips and Cardelli 2009). Designs can be analysed using a variety of techniques at the intermediate level, in technology neutral fashion. In common with digital designs, molecular programming languages are *modular*, and can be built from appropriate biocomponents, which are themselves abstractions of certain biological mechanisms, for example molecular motors, local hybridization or self-assembly. Compositionality at the design level is therefore a desirable feature of the abstractions, both at the level of syntax, as well as semantics, which is harder to achieve in presence of stochasticity and hybrid dynamics.

The abstraction challenge: Despite progress made towards modeling well-mixed molecular systems, for example using stochastic pi-calculus, there is an urgent need to develop *quantitative theories* for the different levels of abstraction hierarchy, in order to support the design of *predictable* molecular systems. In particular, we need to develop compiler technology for molecular programming, including design and implementation of intermediate language abstractions, and efficient algorithms for computing approximate abstractions to a specified level of precision.

Specification notations. Since molecular programs are naturally characterised using quantities, for example kinetics, thermodynamics and stochasticity, the specifications that these programs must meet have to reflect these characteristics. For example, a biosensor must detect the given molecule with sufficiently high probability, and a molecular walker will need to guarantee delivery in a specified time interval, while tolerating a predefined failure rate. Stochastic and real-time temporal logics, for example CSL (Baier et al. 2003; Kwiatkowska et al. 2007), can be used to express many such properties for CTMC models, and logics such as MTL and STL for hybrid models. Conventional temporal logic notations also have their uses; for example, we may wish to require that a molecular program reaches some final state. Furthermore, characteristics typical for device engineering, such as safety, reliability and performance, also apply here. For example, for the DNA walker, which may fail to step correctly on to the next anchorage and instead jump to the following one, we may express the probability of finishing correctly at time T by the CSL formula $P_{=?}[\, F^{[T,T]}\, \text{finished-correct}\,]$.

The specification challenge: Designs must meet specifications that are set in advance. There has been little work concerning *quantitative* specification formalisms that capture aspects specific to the molecular programming setting, such as kinetic energy and thermodynamics, as well as behavioural specifications for out-of-equilibrium systems, e.g. oscillations. Devising suitable logic formalisms to support these more expressive specifications is desirable.

Analysis methods. A broad range of analysis methods exist to exercise models of molecular programs, which are dependent on the modelling framework used. They include techniques similar to those for digital systems, for example *equivalence checking* and *substitutivity* for components, and must extend to capture specialised features, to mention support for DNA self-assembly and structure forming, where thermodynamics and sequence design play a part. For ODE or hybrid models of molecular programs, *analytical* methods or *numerical simulation* can be used to plot average quantities, such as population sizes, over time.

For discrete stochastic models, *stochastic simulation*, for example Gillespie's algorithm, generates individual trajectories by applying Monte Carlo techniques. In contrast, *automated verification* via model checking is able to establish whether a given temporal logic property – e.g., can the system reach a terminal (deadlock) state? – holds in the model. For discrete stochastic models, we apply *automated, quantitative verification* (also known as probabilistic or stochastic model checking) (Kwiatkowska et al. 2007), which accepts a model description and a property specified as a probabilistic temporal logic formula, and computes the probability that the property is satisfied in the model, or expected cost/reward. Compared with simulation, such methods are *exhaustive* and can offer *guarantees* on reliability or performance. The computation can be exact, involving numerical algorithms based on uniformisation (Baier et al. 2003) or fast-adaptive uniformisation (Dannenberg et al. 2013a) (for transient probability), or approximate, based on probability estimation of the proportion of simulated trajectories that satisfy the property (Younes et al. 2006) (referred to as *statistical* model checking). These techniques have been applied to analyse molecular signalling networks (Heath et al. 2008) and have since been adapted to molecular programs. For example, an undesirable deadlock state was automatically discovered in the Cardelli transducer gate design modelled in Visual DSD, and the probability of reaching deadlock obtained using the `PRISM` (Kwiatkowska et al. 2011) probabilistic model checker as a back-end (Lakin et al. 2012). In Dannenberg et al. (2013b,a, 2014), DNA walker systems were subjected to comprehensive analysis of their reliability and performance; a CTMC model was developed based on experimental data (Wickham et al. 2011) and analysed with `PRISM`. The results of the analysis by `PRISM` can be used by the designer to improve the design of the circuit.

Example 2. *We illustrate the role of* automated, quantitative verification *techniques in molecular programming through assessing the reliability and performance of a biosensor implemented using DNA walker circuits (Dannenberg et al. 2013b, 2014). Experiments (Wickham et al. 2011) demonstrate that the walker can traverse a track with one or more omitted anchorages, which shows that the walker is capable of stepping across distances that are double or triple the normal anchorage-to-anchorage distance. This has been incorporated in the model, resulting in the walker being able to jump over blockades, or even reverse direction, which can prevent it from reaching the intended absorbing anchorage and quenching the fluorophore. The walker may also deadlock before reaching an absorbing anchorage, which happens when no uncut anchorages are within reach. This affects the* safety *of biosensors implemented using DNA walkers.*

The trade-off between reliability and deadlock as a function of blockade length is depicted in Fig. 3, obtained by model checking the model in Fig. 1 against the

CSL property $P_{=?}[\,F^{[T,T]}\,\text{end-node}\,]$ that queries the probability of the walker being either finished or deadlocked at time T, where T is 8 mins multiplied by circuit depth. Note that the probability that the walker arrives at the incorrect end-node drops off, while the probability of deadlock increases with the depth of the circuit.

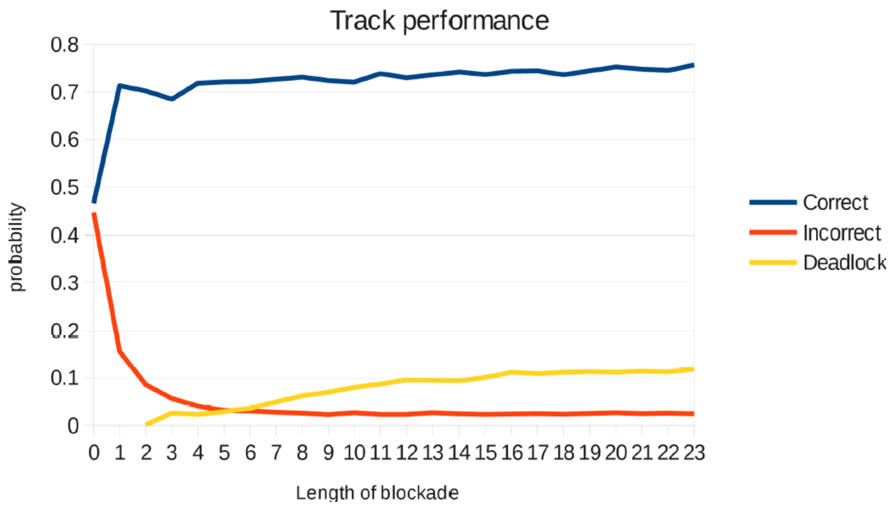

Figure 3: Probability of reaching an absorbing anchorage or deadlock by time T (8 mins times circuit depth) for the walker circuit in Fig. 1.

The analysis challenge: Stochastic simulation methods are known to be computationally intensive and their performance is poor for molecular systems that we wish to design and analyse today. We need much more *efficient* simulation techniques, for example those based on multi-scale simulation which have shown promise. Formal verification techniques, such as automated verification via model checking or interactive theorem proving, are able to establish, via a systematic exploration of the model or mathematical proof, the correctness of a molecular program. Their limitation is the size of the state-space of the model that can be handled, and therefore *scalable, quantitative verification* techniques are a major goal of this research. Promising techniques include SAT/SMT methods, abstraction-refinement schemes and compositional proof methods. Since the models of molecular programs typically include quantitative and possibly continuous aspects, for example stochasticity and energy, we ultimately need the analysis methods to extend to the full class of stochastic hybrid systems. To facilitate their analysis one must apply abstractions and (numerical) *approximations*. This raises the question of the level of precision, including accuracy and error bounds, that the analysis method can inherently guarantee, in turn impacting the predictability of the molecular program's behavior.

Synthesis methods. In addition to being able to perform verification of molecular programs against requirements, an important question is whether it is possible, given a specification, to automatically synthesize a program that is guaranteed to satisfy the specification. This approach would ensure *correct-by-construction* designs, and is in its infancy, particularly regarding *quantitative* synthesis. Techniques developed in systems biology to infer models from experimental data are naturally applicable here. For example, *parameter* synthesis (or estimation) can be used to fit the kinetic rates in a molecular program to observations (Hoops et al. 2006), or even find the optimal values of parameters to satisfy a given quantitative formula for stochastic models (Brim et al. 2013). One example based on the DNA walker circuits is finding the range of walker failure rate parameters so that a specified reliability of the design can be guaranteed. More generally, for a given (quantitative) specification, synthesis methods can be used to automatically configure a system from components; to synthesise a program (mechanism) that guarantees the satisfaction of the property; or even to *evolve* such as program, using techniques from evolutionary computation or genetic programming. For nanorobotic systems, a natural question is whether we can synthesise programmable *controllers* that ensure the safety of the molecular device. Similarly to digital systems, synthesis algorithms are also necessary to support and optimise the *structural* designs, including 2D/3D origami structures and the geometric layout of DNA walker circuits.

The synthesis challenge: This topic has been little researched in the context of quantitative or hybrid models, and its complexity and intractability pose a huge challenge. Promising technique might include template-based synthesis of mechanisms, and developing controller synthesis methods, including from multi-objective specifications.

Integration. A major challenge is to integrate all abstraction levels (from thermodynamics, to sequence, to CRNs), and to achieve fully automatic compilation from high-level specifications to physical structures, with analysis enabled at each level and connected across levels. This can be achieved by relying on modular designs and substitutivity of component specifications for their implementations. The compositional design methodologies and CAD tools that result from this effort will support effective processes to engineer systems from independently specified bio-components.

The integration challenge: Once the integrated CAD tools have been developed, their usefulness must be evaluated on real molecular programs and synthetic biology designs. Criteria for success include the rate of take up of the technologies, improvement in scale and complexity of the designs, the efficiency of software tools, the accuracy of predictions for quantitative aspects, and the quality of the synthesised designs.

4 Conclusions

This paper has given a brief overview of the emerging field of molecular programming, discussing existing techniques to support the design process and outlining future research challenges. Molecular programming has the potential to revolutionise personalised medicine and synthetic biology, with many applications envisaged, for example programmable intraveneous systems to deliver drugs that target specific molecules. Since safety is a paramount concern when deploying such devices, we have put forward the view that quantitative, automated verification techniques will constitute a key component of design automation for molecular programming. This can only be achieved through collaboration between experimental scientists, engineers and computer scientists.

Acknowledgments. The author is supported by the ERC Advanced Grant VERIWARE and would like to acknowledge participants of the NSF workshop “Advances in Molecular Programming and Computing: Toward Chemistry as a New Information Technology” held in 2013 in Copenhangen for helpful discussions.

References

D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

N. Aubert, C. Mosca, T. Fujii, M. Hagiya, and Y. Rondelez. Computer-assisted design for scaling up systems based on dna reaction networks. *Journal of The Royal Society Interface*, 11(93):20131167, 2014.

C. Baier, B. Haverkort, H. Hermanns, and J. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on Software Engineering*, 29:524–541, 2003.

L. Brim, M. Ceska, S. Drazan, and D. Safránek. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In N. Sharygina and H. Veith, editors, *Proc. CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 107–123. Springer, 2013.

L. Cardelli. Two-domain DNA strand displacement. *Developments in Computational Models*, 26:47–61, 2010.

L. Cardelli and A. Csikász-Nagy. The cell cycle switch computes approximate majority. *Nature Scientific Reports*, 2, 2012.

Y.-J. Chen, N. Dalchau, N. Srinivas, A. Phillips, L. Cardelli, D. Soloveichik, and G. Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8(10):755–762, 2013.

F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33-34):3065–3084, 2009.

F. Dannenberg, E. M. Hahn, and M. Kwiatkowska. Computing cumulative rewards using fast adaptive uniformisation. In A. Gupta and T. Henzinger, editors, *Proc. 11th Conference on Computational Methods in Systems Biology (CMSB'13)*, volume 8130 of *LNCS*, pages 33–49. Springer, 2013a.

F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. Turberfield. DNA walker circuits: computational potential, design, and verification. In D. Soloveichik and B. Yurke, editors, *Proc. 19th International Conference on DNA Computing and Molecular Programming (DNA 19)*, volume 8141 of *LNCS*, pages 31–45. Springer, 2013b.

F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. Turberfield. DNA walker circuits: computational potential, design, and verification. *Natural Computing*, 2014. To appear.

D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

M. Hagiya. From molecular computing to molecular programming. In A. Condon and G. Rozenberg, editors, *DNA Computing*, volume 2054 of *Lecture Notes in Computer Science*, pages 89–102. Springer, 2000.

J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 319(3):239–257, 2008.

S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI - a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.

C. Jung and A. D. Ellington. Diagnostic applications of nucleic acid circuits. *Accounts of Chemical Research*, 2014. To appear.

T. G. Kurtz. The relationship between stochastic and deterministic models for chemical reactions. *The Journal of Chemical Physics*, 57:2976, 1972. ISSN 00219606. doi: 10.1063/1.1678692.

M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In *Proc. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 449–458. ACM Press, September 2007.

M. Kwiatkowska and C. Thachuk. Probabilistic model checking for biology. In *Software Safety and Security*, NATO Science for Peace and Security Series - D: Information and Communication Security. IOS Press, 2014. To appear.

M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

M. Kwiatkowska, G. Norman, and D. Parker. *Symbolic Systems Biology*, chapter Probabilistic Model Checking for Systems Biology, pages 31–59. Jones and Bartlett, 2010.

M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

M. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, 9(72):1470–1485, 2012.

H. H. McAdams and a. Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences of the United States of America*, 94:814–9, 1997. ISSN 0027-8424.

A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *Journal of the Royal Society Interface*, 2009.

L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332:1196–1201, 2011.

A. Regev, W. Silverman, and E. Y. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pacific Symposium on Biocomputing*, pages 459–470, 2001.

P. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440:297–302, 2006.

G. Seelig, D. Soloveichik, D. Zhang, and E. Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314:1585–1588, 2006.

D. Soloveichik, G. Seelig, and E. Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Science*, 107(12): 5393–5398, 2010.

S. F. J. Wickham, M. Endo, Y. Katsuda, K. Hidaka, J. Bath, H. Sugiyama, and A. J. Turberfield. Direct observation of stepwise movement of a synthetic molecular transporter. *Nature nanotechnology*, 6:166–9, 2011.

E. Winfree. Toward molecular programming with DNA. *SIGPLAN Not.*, 43(3): 1–1, Mar. 2008. URL `http://doi.acm.org/10.1145/1353536.1346282`.

P. Yin, H. Yan, X. G. Daniell, A. J. Turberfield, and J. H. Reif. A unidirectional DNA walker that moves autonomously along a track. *Angewandte Chemie International Edition*, 43:4906–4911, 2004.

H. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(3):216–228, 2006.

B. Yurke, A. Turberfield, A. Mills, F. Simmel, and J. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406(6796):605–8, 2000.

D. Zhang and G. Seelig. Dynamic DNA nanotechnology using strand displacement reactions. *Nature Chemistry*, 3:103–113, 2011.

D. Y. Zhang, A. J. Turberfield, B. Yurke, and E. Winfree. Engineering entropy-driven reactions and networks catalyzed by DNA. *Science*, 318(5853):1121, 2007.