ACS SyntheticBiology

pubs.acs.org/synthbio

Research Article

Toward Full-Stack *In Silico* Synthetic Biology: Integrating Model Specification, Simulation, Verification, and Biological Compilation

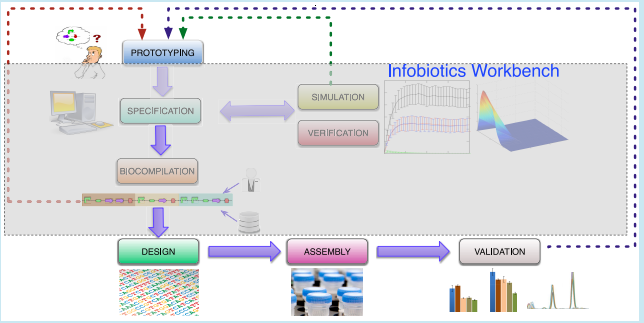
Savas Konur,* Laurentiu Mierla, Harold Fellermann, Christophe Ladroue, Bradley Brown, Anil Wipat, Jamie Twycross, Boyang Peter Dun, Sara Kalvala, Marian Gheorghe, and Natalio Krasnogor*

Cite This: *ACS Synth. Biol.* 2021, 10, 1931−1945

Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: We present the Infobiotics Workbench (IBW), a user-friendly, scalable, and integrated computational environment for the computer-aided design of synthetic biological systems. It supports an iterative workflow that begins with specification of the desired synthetic system, followed by simulation and verification of the system in high-performance environments and ending with the eventual compilation of the system specification into suitable genetic constructs. IBW integrates *modeling*, *simulation*, *verification*, and *biocompilation* features into a single software suite. This integration is achieved through a new domain-specific biological programming language, the Infobiotics Language (IBL), which tightly combines these different aspects of *in silico* synthetic biology into a full-stack integrated development environment. Unlike existing synthetic biology modeling or specification languages, IBL uniquely blends modeling, verification, and biocompilation statements into a single file. This allows biologists to incorporate design constraints within the specification file rather than using decoupled and independent formalisms for different *in silico* analyses. This novel approach offers seamless interoperability across different tools as well as compatibility with SBOL and SBML frameworks and removes the burden of doing manual translations for standalone applications. We demonstrate the features, usability, and effectiveness of IBW and IBL using well-established synthetic biological circuits.



As DNA sequencing and synthesis technology become cheaper and more easily accessible,[1] the scale and complexity of engineering biology projects is set to grow. Furthermore, rapidly converging biotechnology and computing science are accelerating the adoption of synthetic biology across scientific disciplines as well as industrial applications. Some of the rapid progress being made include advances in genome editing *via* CRISPR-Cas9 variants (*e.g.*, refs 2, 3), biomedical materials design,[4] biofilms engineering as nano-factories,[5] synthetic biology routes to functional materials,[6] environmental remediation,[7] the engineered transformation of an *E. coli* strain so it can manufacture all its carbon biomass directly *via* CO_2 consumption,[8] novel biomedical sensors,[9] to the creation of *in vivo* DNA-based memory devices.[10] These advances have also prompted progress in computer aided biology, ranging from computational simulations,[11−14] to metabolic engineering optimization tools,[15] new domain specific languages from describing engineered circuits,[16] to new biological barcodes for version control systems of living cells.[17] Along with rapid developments in this field, a number of software suits for modeling and analyzing synthetic biology have been developed. Among these tools are iBioSim,[18] Cello,[19] Gro,[20] Tinkercell,[21] Eugene,[22] and Proto.[23]

Yet, contemporary progress in synthetic biology is largely driven by a cumbersome trial and error cycle in the laboratory. Various computational tools, which include computer aides for designing and analyzing synthetic biological circuits, have been released to assist this development process. These tools commonly support a specific functionality, such as genetic design specification, sequence specification, or simulation. For example, Gene Designer,[24] Eugene,[25] GenoCAD,[26] GEC,[27] MoSeC,[28] and PartsGenie[29] allow a user to express, and in part compile, genetic sequence designs. Other tools, such as COPASI[30] and VCell,[31] support biomolecular simulation. Interoperability between these tools is often impractical due to the lack of a common file format able to capture the information required by each individual tool. There have been attempts to capture their common semantics through standard meta-languages such as SBOL[32] and SBML,[33] but it is difficult to reverse-translate the low-level description obtained from one tool into the input required by the next. Full interoperability, while a laudable task, is very difficult to

Received: April 6, 2021
Published: August 2, 2021

achieve. The Clotho toolset was an attempt to create a fully integrated set of app-style tools,[34] but the software development difficulties for such a project are very significant. When taking biological constructs from idea to working prototype through a series of refinements, it becomes increasingly harder to maintain the various formal specifications while ensuring that each of them is up-to-date and consistent with every other object models.

Because of the difficulty in interoperability, synthetic biologists are typically forced to either dispense with some of the available tool support or work with a set of independent formalizations of their synthetic design specifications—in which case crucial details may be lost.

In this paper, we present the *Infobiotics Workbench* (https://infobiotics.org), a computer-aided design suite for synthetic biology that assists synthetic biologists in an informed, iterative workflow of system specification, verification, simulation, and biocompilation. The Infobiotics Workbench (IBW) features a unique domain-specific Infobiotics Language (IBL), which integrates these different computational aspects into a single specification file. Unlike conventional approaches, IBL defines a simple combined grammar for modeling, verification, and biocompilation statements, which would otherwise require using complex formalisms and sophisticated transformations for utilizing independent tools and performing different computational analyses. This novel approach offers seamless interoperability across different tools as well as compatibility with SBOL and SBML frameworks and removes the manual translation requirement for standalone applications.

IBW replaces an older version.[35] It now presents several new state of the art features, including (i) an intuitive and expressive synthetic biology design language, (ii) a simulation component that implements all the variants of Gillespie's stochastic simulation algorithms (SSA), (iii) a prediction tool that selects the best performing SSA using machine learning algorithms, (iv) a parallel implementation of Gillespie algorithms executed on cloud GPU clusters, (v) a verification component that allows verifying queries and design requirements using natural language queries, (vi) a biocompilation module that allows automated compilation of a specified synthetic circuit into eventual genetic sequence information and (vii) import from/export to standard data exchange formats, *e.g.*, SBOL and SBML. This makes IBW a unique platform that integrates these unique features. A comprehensive comparison of IBW with the existing tools is provided in Table S1 of the Supporting Information.

The workbench adopts well-established design principles that guide both experienced and nonexperienced biologists in refining a putative functionality into a formally specified document for fabricating a nucleotide sequence after undergoing some computational analysis.

IBW's unique features are underpinned by strong theoretical foundations spanning advanced stochastic simulation algorithms, high-performance computing, formal verification, ontology languages, biological property mining, constraint satisfaction algorithms, and machine learning.

RESULTS AND DISCUSSIONS

The Infobiotics Language. Infobiotics Workbench (IBW) relies on the Infobiotics Language (IBL), a domain-specific language based on synthetic biology. IBL is very unique in the sense that it integrates all computational tasks into one single language. Namely, it gives directives that indicate what simulations to run, what verifications to calculate, and what synthetic circuits to biocompile. Integrating these directives into the same domain-specific language enables us to fully support an engineering-inspired workflow of iterative design, simulation, verification, and compilation of synthetic biological systems.

IBL has used terms from ontologies available in the literature, *e.g.*, SBOL[32,36] and SyBiOnt[37] as well as Gene Ontology[38] and Sequence Ontology.[39] The language has also been complemented with particular terms that are tailored to the potential user base such as RULEs to capture chemical reactions and PROCESSes to reuse (groups of rules) and provide a more abstract representation (and decomposition) of complex processes (see Figure 1).

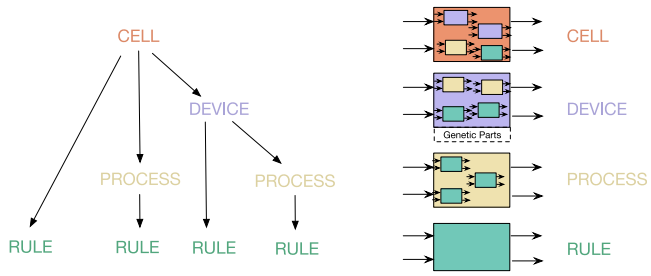


Figure 1. Hierarchical representation of biological entities. A RULE represents a chemical reaction; a PROCESS is a group of rules and provides a more abstract representation (and decomposition) of complex processes; a DEVICE refers to an assembly of genetic parts and biological building blocks; a CELL represents a bacterial cell.

IBL provides statements to declare biological entities and their interactions along with statements to (a) annotate PROCESSes with stochastic rates for stochastic simulation; (b) annotate PROCESSes, DEVICEs, and CELLs with verification statements to be verified; and (c) annotate GENEs, PROMOTERs, and other biological entities with sequence information or links to online repository entries required for biomatter compilation. IBL further allows the annotation of designs with custom information for use-cases not directly supported by the core language.

IBL permits a robust encapsulation and hierarchal structure of biological entities (see Figure 1). The language has the following structure:

- Molecular species represent any entity, from PROTEIN to DNA to MOLECULE.
- RULEs are used to define a transformation, most often a chemical transformation.
- PROCESSes are collections of RULEs. They make it easier to reuse rules and typically represent a more abstract process, *e.g.*, constitutive protein expression.
- DEVICEs are collection of PROCESSes and RULEs. They refer to device in the synthetic biology sense: a piece of DNA made up of parts.
- SYSTEMs contain DEVICEs and RULEs.
- PLASMIDs are collections of SYSTEMs, DEVICEs and RULEs. They are useful when more than one type of plasmid is used in the cell.
- CHROMOSOMEs are collections of SYSTEMs, DEVICEs, and RULEs.
- CELLs refer to biological cells. They are a collection of PLASMIDs, SYSTEMs, DEVICEs, PROCESSes, and RULEs.

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

- REGIONs contain CELLs, PROCESSes, RULEs, and molecular species.

IBL allows the user to declare common biological parts, including PROMOTER, GENE, RIBOSOME, TERMINATOR, CLONING SITE, DNA, RNA, and RBS with optional declaration of sequences either in online repositories or in a built-in database as well as nongenetic molecular components such as PROTEIN and MOLECULE (see Algorithm 1). Thus,

```
1 define ConstitutiveProteinExpression typeof PROCESS {
2     promoter = PROMOTER (URI="http://biobricks.org/...")
3     gene = GENE (URI="http://parts.igem.org/...")
4     protein = PROTEIN ()
5     // some rules ...
6 }
```

Algorithm 1: IBL specification of biological parts and non-genetic molecular components.

declared biological parts can be connected to each other *via* reaction rules, which can be declared as either reversible or irreversible. Since complexification is particularly important in biological systems, *e.g.*, in the quaternary structure of proteins or genetic regulation, complexes can be used in rules without the need to explicitly declare them.

To ease the integration of experimental knowledge, IBL supports specification of *concentrations* and *reaction rate constants* in SI units and automatically converts between units where possible. Concentrations can be specified in *molar* (*i.e., mol/liter*), *millimolar*, *micromolar*, and *nanomolar* units. Alternatively, concentration can also be specified as an amount of molecules, which can be interconverted with molar concentration using a defined cell volume (which defaults to one *femtoliter*). Similarly, reaction rate constants can be specified in the units *per second*, *per minute*, and *per hour* in case of unimolecular reactions, or *per mole per second*, *etc.* for bimolecular reactions.

As an example, consider the regulation of the P*lac* promoter by LacI. The IBL segment shown in Algorithm 2 first declares the LacI protein and its reversible dimerization reaction (lines 1−4), followed by the promoter and the reversible regulator binding (lines 5−8).

```
1 LacI = PROTEIN (concentration=10 uM)
2 RULE lacI_dimerization: LacI + LacI <-> LacI~LacI
3 lacI_dimerization.forwardRate = 1e0 M-1 min-1
4 lacI_dimerization.backwardRate = 125 min-1
5 Plac = PROMOTER ()
6 RULE lac_repression: Plac + LacI~LacI <-> Plac~LacI~LacI
7 lac_repression.forwardRate = 1e0 min-1
8 lac_repression.backwardRate = 10 min-1
```

Algorithm 2: IBL specification of two reversible reactions involving the LacI repressor and the P*lac* promoter. Line 1 defines LacI to be a protein present in 10 micromoles per liter concentration; line 2 declares the reversible dimerization of two LacI units, indicated by the tilde; lines 3-4 annotate this reaction with forward and backward rate *constants*; line 5 defines Plac to a promoter (present with one molecule by default); line 6 declares the reversible binding of the LacI dimer to the promoter; lines 7-8 annotate the respective rate constants.

To support scalable, modular, and reusable designs, IBL allows for the declaration of PROCESS, which groups together a set of rules involving abstract biological entities (see Algorithm 1). Once defined, processes can be instantiated for specific components. For example, one can specify a process that collects all of the reactions that constitute general regulated gene expression, which then can be instantiated for different transcription factors and genes. In this manner, processes introduce a layer of modularity and abstraction similar to function declarations in imperative programming languages. This is particularly convenient in conjunction with an "*import*" statement that allows references to process definitions across files. In this manner, general purpose libraries for recurrent biological processes can be built and later reused to declare specific biological circuits with relatively little effort.

In addition to PROCESS, IBL also provides the means to group molecules, rules, and process instantiations into DEVICE (see Algorithm 3) to capture the action of an entire genetic device, *i.e.*, a strand of DNA containing several promoters, genes, and their encoded proteins.

```
1 operon = DEVICE (
2           parts =[PnahR,nahR,NahR],
3           input =[SA~NahR],
4           output =[NahR]
5           ){
6           PROCESS p1 = ConstitutiveProteinExpression(promoter = PnahR,
             gene = nahR, protein = NahR)
7           // some rules ...
8           }
```

Algorithm 3: IBL specification of a device using an instance of the process defined in Algorithm 1.

In addition to logical encapsulation, IBL supports physical encapsulation of molecules, processes, and devices into CELLs, as well as colocation of molecules, rules, processes, and cells into REGIONs. By default, cells and regions physically separate molecules by introducing boundaries. However, IBL supports mechanisms to define translocation of molecules from cells into regions and *vice versa*, which allows for expressing secretion and uptake processes as well as cell-to-cell communication. Thus, IBW can model populations of potentially mixed bacterial cohorts. Algorithm 4 gives an example.

```
1 DEFINE myCell TYPEOF CELL {
2     AHL = MOLECULE ()
3     RULE ahl_permeation: AHL <-> OUTSIDE
4 }
5 DEFINE well TYPEOF REGION {
6     AHL = MOLECULE (concentration=100 mM)
7     cell = myCell()
8 }
```

Algorithm 4: IBL specification of a cell residing in a region that contains the signalling molecule AHL. The cell defines a membrane permeation process for AHL.

IBL also allows incorporating verification and biocompilation directives into various compartments, *e.g.*, devices, cells and processes, to make sure that certain constraints are met at the design stage. For example, the following verification and compilation statements in Algorithm 5 can be added within the device defined in Algorithm 3.

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

```
// verification rules
VERIFY [NahR > 1 uM] EVENTUALLY HOLDS WITHIN 10 s
// bio-compilation rules
ATGC ARRANGE: nahR, PnahR
```

Algorithm 5: Some verification and bio-compilation directives.

The Infobiotics Workbench. IBW (https://infobiotics.org) provides an *in silico* laboratory for synthetic biology and a single, integrated, user-friendly, computer aided design platform for Synthetic Biology parts and devices.

A screenshot of the IBW user interface is shown in Figure 2. IBW provides support for the specification of synthetic devices, their simulation, verification, and biomatter compilation.

Simulation. IBW features two types of simulation environments.

CPU-Based Simulation. *NGSS*[40] is a high performance CPU-based simulation environment, which implements eight exact and one approximating variant of Gillespie's stochastic simulation algorithm (Direct Method (DM), Optimized DM, Sorting DM, Logarithmic DM, Partial Propensity DM, Next Reaction Method, First Reaction Method, Composition Rejection, and Tau Leaping).

Depending on the simulated reaction network, runtime performance of these algorithms was found to differ by orders of magnitude with no single winning strategy that would outperform competitors on all model systems. Using machine learning techniques, we are able to predict the best performing simulation algorithm for a given reaction network.

Our method relies on representing stochastic models as graphs of dependencies between reactions or species. Using these graphs, we are able to build a topological profile of each model (*e.g.*, number of vertices and edges, graph density, graph degree *etc.*). Machine learning algorithms can learn how these topological and graph-theoretic features of the underlying network of a model affect the simulation time of SSAs by analyzing the performance (reactions executed per second of CPU time) of each algorithm measured using benchmark models (obtained from the BioModels database[41]). The predictor performs a model topology analysis and uses the results to predict the fastest SSA for that model.

We have implemented this method as a performance benchmarking suite, *SSA Predictor*,[13] which allows for a direct and unbiased comparison of stochastic simulation algorithms. SSA Predictor is fully integrated in IBW. So, when a model is simulated, IBW automatically chooses the fastest predicted SSA algorithm, and performs the simulation using this algorithm.

Apart from being implemented for best performance, the simulator also employs MPI to distribute multiple simulation runs on computing clusters and OpenMP for multicore systems.

GPU-Accelerated Simulation. IBW also integrates a GPU parallel implementation of the Gillespie SSA, taking advantage of the CUDA platform. The implementation is designed to run in an HPC environment, with which IBW communicates for submitting simulation requests and retrieving results. The performance of the GPU implementation of the Gillespie SSA mainly comes from computing the species evolutions for all the runs, in parallel, for each individual time point. Moreover, it also takes advantage of the parallel implementation for other different internal processes from the Gillespie SSA workflow, such as using the Hillis Steele algorithm[42] for computing the propensity prefix sums.

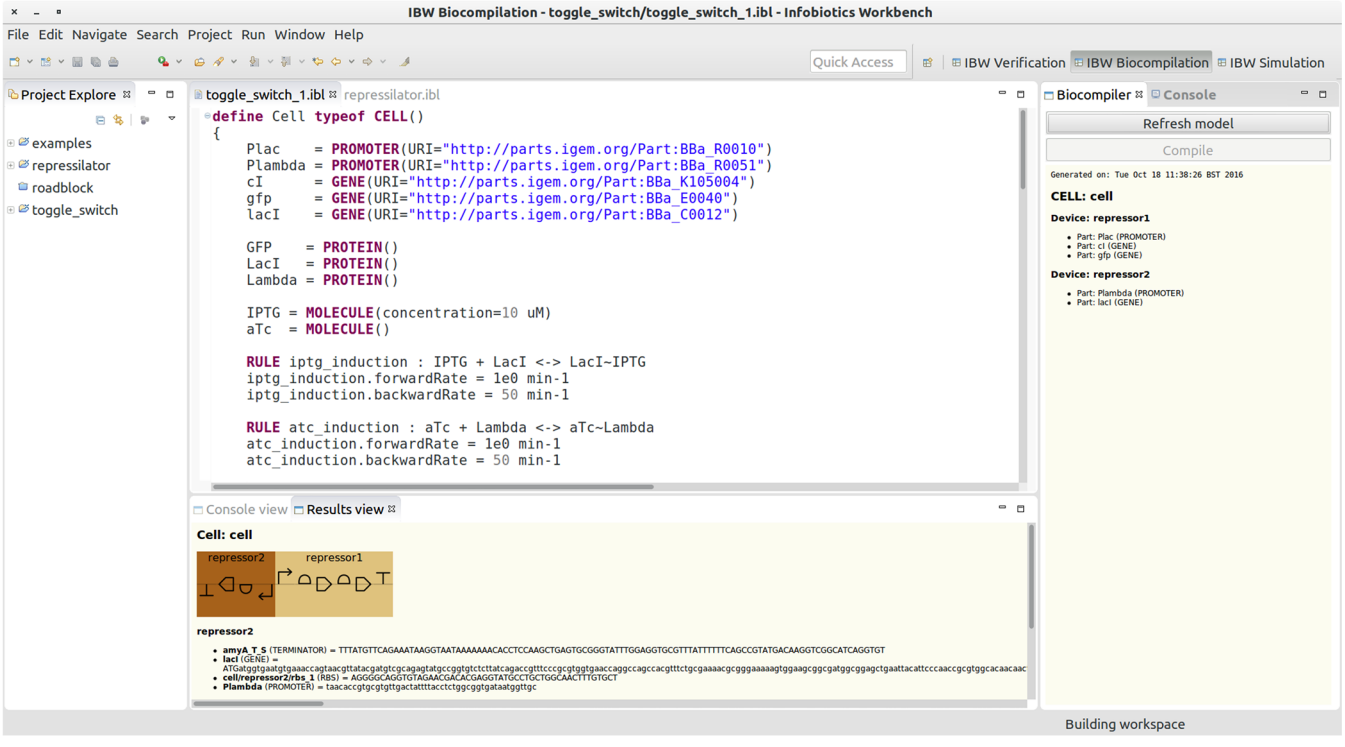


Figure 2. A screenshot of IBW in its biocompilation perspective. The integrated development environment features a project navigator (left), source code editor (top center), biocompilation controller (right) and compilation result window (bottom center). The toolbar and menu provides access to typical development features including refactoring and collaborative versioning tools such as git.

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

GPU provides significant performance gains compared to the CPU counterpart. As Figure 3 suggests, the GPU simulator provides a significant performance gain compared to the CPU simulator.

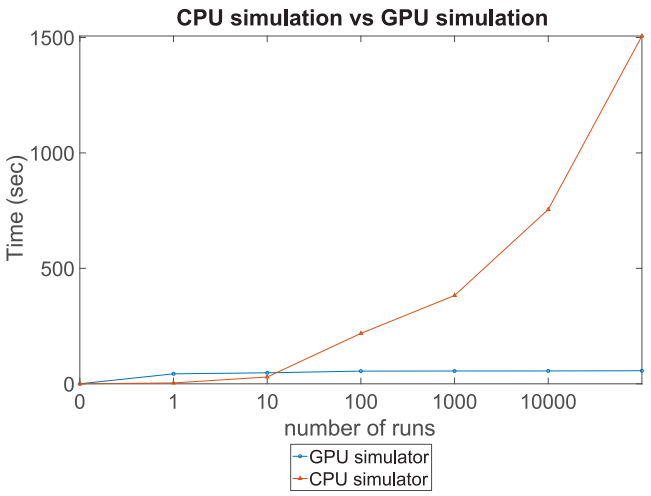


Figure 3. Performance benchmark comparison of IBW's CPU and GPU simulators using the quorum sensing model (see Quorum Sensing section). The GPU simulator provides a much faster and efficient alternative compared to the CPU simulator (implementing serial algorithms) as it uses parallel algorithms, run in high performance environments.

Verification. Designing biological systems can be complex, and molecular interactions that could disrupt the entire system's behavior are easy to miss. To guard against such design flaws, IBW provides the means to annotate designs with verification statements that the behavior of the specified circuit must satisfy.

Formal verification describes a family of techniques for exhaustively analyzing the logical correctness of a system. The essence of formal verification is analyzing a logical requirement (typically stated in temporal logic[43,44]) against all possible behaviors of the system in question. Formal verification is a well-established research field, and it has many applications, *e.g.*, safety-critical systems,[45] concurrent systems,[46] distributed systems,[47] network protocols,[48] stochastic systems,[49] multi-agent systems,[50,51] pervasive systems,[52−54] swarm robotics,[55] and biology[56] as well as some engineering applications.[57−59]

Formal verification is used to validate the system correctness in relation to a desired functionality. In conventional validation methods, the analysis of system behavior mainly relies on "simulation". However, many important system properties cannot be inferred using simulation. Also, simulation does not guarantee that the system is error-free and reliable because simulation can only show "presence of errors, not their absence".

Verification statements allow checking for high-level properties and are thus comparable to unit and integration tests in software development. By integrating verification in the core IBL specification, we ensure that test cases can be defined at or near the point of process definition. This prevents potential divergence of specification and verification, which can easily occur in continuous development.

Commonly, verification statements are expressed using temporal logics.[60−62] However, devising correct temporal logic clauses for an intended verification statement can introduce an undesired layer of complexity. To ease verification, IBL therefore allows the user to express statements as text-based natural language narratives.

The verification component has another important feature. It provides a set of so-called *property patterns* based on the most frequent properties in biology.[56,63] The idea is to categorize most recurring properties into a pattern system by defining a generic representation of instances of numerous properties in the literature.

To identify the most relevant and frequent patterns, we have extensively surveyed the case studies addressing the formal analysis of biological and biochemical systems and derived the property patterns for a variety of common verification scenarios. In Algorithm 6, we provide some example patterns currently implemented in IBW.

```
1 VERIFY [ GFP > 0 uM ] EVENTUALLY HOLDS
2 VERIFY [ GFP > 0 uM ] EVENTUALLY HOLDS WITHIN [0,10] s WITH
    PROBABILITY > 0.9
3 VERIFY [ GFP > 0 uM ] NEVER HOLDS
4 VERIFY [ GFP > 0 uM ] ALWAYS HOLDS
5 VERIFY [ GFP > 0 uM ] HOLDS INFINITELY OFTEN
6 VERIFY [ GFP > 0 uM ] HOLDS IN STEADY-STATE
7 VERIFY [ GFP > 0 uM ] HOLDS UNTIL [ SA-NahR > 0 uM ] HOLDS
8 VERIFY [ SA-NahR > 0 uM ] IS FOLLOWED BY [ GFP > 0 uM]
```

Algorithm 6: IBL's verification statements.

Users can integrate verification in their design workflow by instantiating any of these verification statements (presented in Algorithm 6) in IBL models. By tweaking the variables in these patterns they obtain the desired system *properties* that are required to be satisfied. The verification results indicate that the model either satisfies the properties or that it violates them. A model does not satisfy the properties if the model has an error (*e.g.*, sufficient amount of proteins cannot be produced) or the design is faulty (*e.g.*, certain proteins are never produced). For the former case, the user should refine the model parameters and then repeat the entire process. For the later case, the verification results can be considered as a counterexample and used to revise the design of the system.

IBW automatically translates the verification statements into the syntax of the corresponding temporal logics, which is then delegated to model-checking tools for exhaustive formal verification. The type of property pattern thereby dictates the choice of the verification tool. Currently, IBW interfaces with both nonstochastic—NuSVM[64]—and stochastic—PRISM,[65] MC2[66]—model-checking tools to check *qualitative* and *quantitative* properties, respectively. Stochastic model checkers are useful to verify properties about the *likelihood* of the observation of certain behavior, whereas nonstochastic model checkers are useful when quantitative analysis might not be needed if, for example, we only want to observe the detection of molecular species rather than measuring their concentration. In such cases, we can only rely on qualitative analysis, where we can apply some abstraction methods, *e.g.*, removing kinetic constants from stochastic models, to reduce the model complexity.

Although formal verification is very useful for system analysis, it has a major drawback: *state explosion* problem,[67] meaning that the state space grows so quickly that model checking cannot stay scalable. This is especially the case for

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

large models (very common in biology) because model checkers search this large state space exhaustively.

IBW adopts the best practices and new research developments in this field to make the verification process much faster and very efficient for the users. Unlike safety critical systems, where even one in a billion chance of system failure cannot be allowed, biological systems are more tolerable to approximate results based on a high confidence value. So, instead of using *analytic* methods that calculate *precise* results, we can use *statistical* methods that provide *approximate* results based on a confidence interval. For this reason, when performing verification, wherever possible, IBW uses *statistical model checking*.[68] The core idea of statistical model checking is to simulate the system for finitely many runs according to the distribution defined by the system, and use hypothesis testing to infer whether the samples provide a statistical evidence for the satisfaction or violation of the specification.[69] Since the state space is explored partially, the performance is increased. Due to its advantage on scalability, statistical model checking offers much faster results.

We improve statistical model checking further by taking advantage of our GPU parallel implementation of the Gillespie SSA running in an HPC environment. The significant performance gain obtained in our simulations reported in the Simulation section is also utilized in the verification process.

IBW reduces the verification time even further by allowing the user to constrain verification statements to system submodules, including individual cells, devices, and processes. This can be done by simply placing the verification statement in the cell, device, or process in question. Verification will then be performed only for the submodule that the statement occurs in. This provides significant performance gains compared to conventional approaches, where verification is applied to the whole system.

All these state of the art solutions make IBW one of the fastest and the most scalable tools available in the verification of large biological systems.

Biocompilation. Subsequent to successful verification and simulation, the design must be mapped to specific nucleotide sequences that can be produced and incorporated into host cells. IBW allows for automated compilation of a specified synthetic circuit into eventual genetic sequence information by interfacing with the *ATGC*[70] (Assistant To Genetic Compilation) component. With no user intervention, ATGC: (a) completes abstract device specifications by adding ribosome binding sites (RBS), spacers and effective terminators, (b) finds a viable arrangement of the parts by solving an equivalent integer optimization problem, and (c) finds the parts' sequences either in online repositories (*e.g.*, iGem) or in a built-in database, populated by parts from Biofab[71] and REbase.[72] Unless explicitly stated, RBS sequences are generated with Salis' RBS calculator[73] and a user-specified initiation translation rate. ATGC thus automatically produces a biologically plausible sequence reflecting the original aim of the user. A number of ATGC directives allow the user to control its output. The types of constraints available at present (to increase in the future) were directly inspired by situations encountered by experimentalists. For example, one such directive controls, the automatic insertion of cloning sites. Since cloning sites are noncoding sequences, they are typically not present in high-level specifications, which primarily model the dynamics of gene expression. Instead, the user employs an ATGC directive that inserts a placeholder for a restriction site whose sequence is calculated by the biocompiler from a large curated database of restriction enzymes. The biocompiler ensures that the chosen cloning site sequence does not appear in other parts of the full sequence. In particularly constrained situations, ATGC will perform codon replacement in order to find a solution. Other directives allow existing DNA fragments, which may already contain several components, to be reused. The tool takes into consideration the bidirectionality of double-stranded DNA, which allows certain devices to be present in each of the two strands. This mimics the optimization present in natural devices.

Compatibility with Other Standards. After an IBL model is optionally biocompiled, the user has the option of exporting it as SBOL (supporting two modes of export one with compiled sequences and one without)[32] or SBML,[33] which are both compatible with a growing number of synthetic biology softwares. For example, j5[74] can take an SBOL input and produce an assembly protocol for the construct, which is typically the next step after designing the DNA sequence. In addition to the export feature, IBW can also import from SBOL and SBML, generated by other tools, *e.g.*, iBioSim[18] and Cello.[19]

The import IBW functionality takes as input an SBML/SBOL model and translates its different entities into their equivalent IBL representation. The translation from SBML/SBOL to IBL is implemented in Java and features the JSBML (https://github.com/sbmlteam/jsbml) and LibSBOLj (https://github.com/SynBioDex/libSBOLj) libraries for parsing SBML and SBOL models, respectively. Upon parsing, the corresponding abstract syntax tree of the input model is traversed and the corresponding semantic entities are converted into their IBL counterparts.

In a similar fashion, the IBW export functionality allows for an IBL model to be exported into an equivalent SBML/SBOL representation. The translation from IBL to SBML/SBOL is also implemented in Java and takes advantage of the JSBML and LibSBOLj. By traversing the internal IBW data structure corresponding to the IBL model, each IBL semantic entity is converted into an equivalent SBML/SBOL semantic (language-agnostic) data structure, which is constructed using Java classes defined in the above-mentioned libraries. Once a semantic SBML/SBOL data structure is obtained, the final SBML/SBOL (XML-formatted) models are generated by taking advantage of the same libraries.

SBOL conversion captures the structural information and hierarchy of the IBL model. This hierarchy (approximately region → cell → molecule/device) is translated into SBOL objects, and if the model is further biocompiled, biocompilation specific information (sequences, cloning sites, and ribosome-binding sites) are also retained. However, SBOL omits numerical information like concentrations, rate constants, compartment sizes, and units. This prevents the SBOL model from being simulated and verified. On the other hand, SBML conversion captures functional aspects of the IBL model, which includes the previously mentioned numerical information. SBML also allows us to retain the structural hierarchy, although biocompilation-specific features like DNA structures cannot be expressed.

IBW targets these shortcomings by making all computational features accessible to the SBOL and SBML communities. We note that IBL is a true programming language to be used by humans to program biology, whereas SBOL and SBML only provide data models, and are not biological programming

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

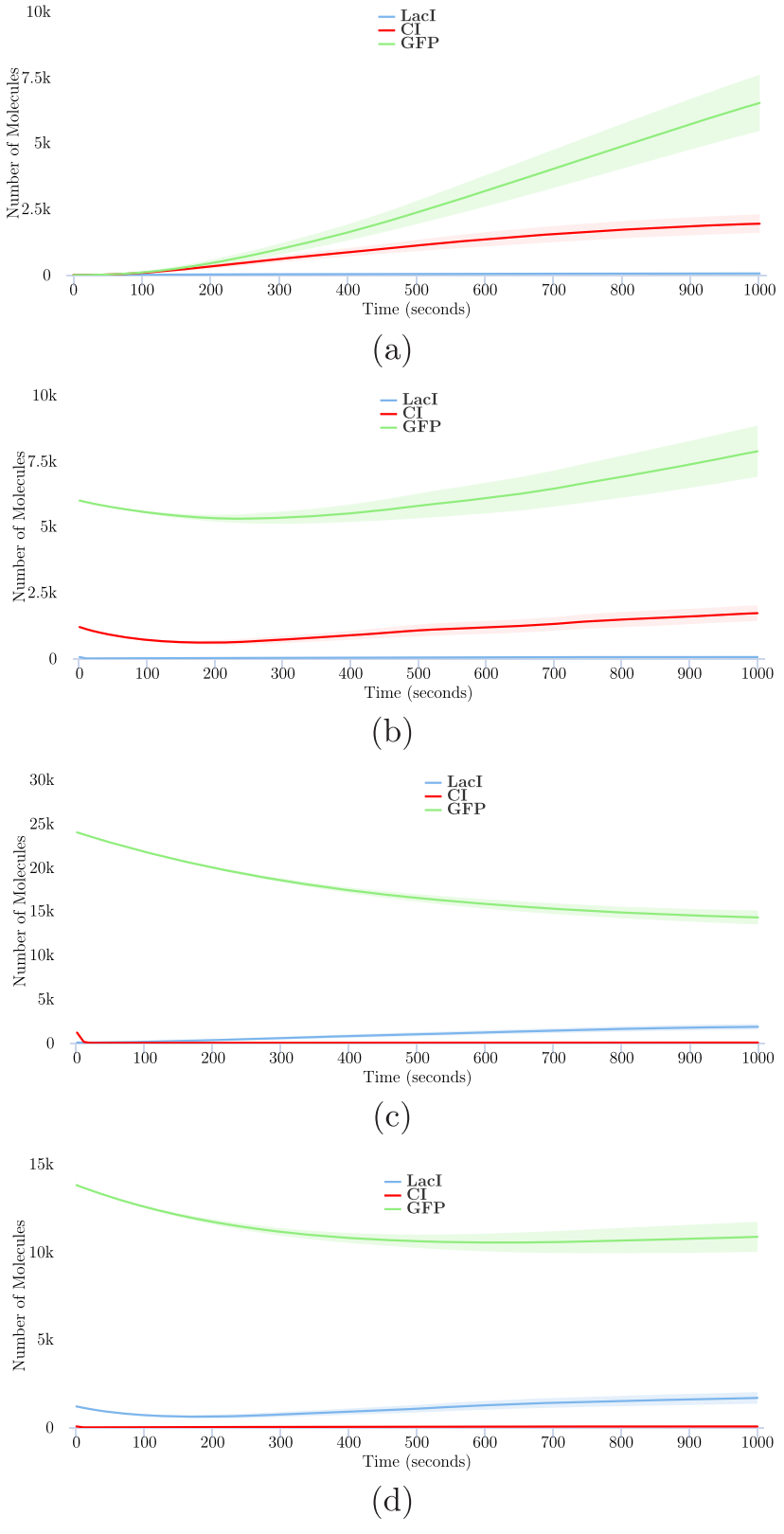


Figure 4. Four sequential stochastic simulations of the toggle switch. In the first simulation, the cell is suspended in IPTG, leading to the activation of the switch and production of GFP (green trace) and CI (red trace). In the second one, CI and GFP production is maintained despite the absence of IPTG. In the third simulation, the cell is suspended in aTc, leading to the deactivation of the switch, production of LacI (blue trace) and decay of GFP. In the final one, the switch resides in its off state despite the absence of aTc. Traces show the mean and standard deviations of 50 simulation runs.

languages. IBL can be compiled down to SBOL and/or SBML for standardized exchange but in doing so information that is currently captured by IBL will be lost, *e.g.*, information about verification or biocompilation constraints.

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

We note that ShortBOL[75] has been recently introduced to generate SBOL files using a human readable language. However, it is not a comprehensive programming language like IBL. It does not capture the functional information (*i.e.*, behavioral models at the molecular level). Also, some unique elements of IBL, *e.g.*, verification and biocompilation statements, cannot be expressed in ShortBOL.

METHODS

Implementation. IBW has been implemented as an Eclipse rich client application using Xtext as a domain specific language framework. It is released under the MIT license and both user and developer IBW editions can be obtained from https://infobiotics.org.

Case Studies. We demonstrate the applicability of IBW and IBL using a number of case studies from the literature: Gardner *et al.*'s "toggle switch",[76] Elowitz and Leibler's "repressilator",[77] Dockery and Keener's "quorum sensor",[78] and Myers' "logic gates".[79] For each case, we discuss the respective IBL specification, along with its verification, simulation, and biocompilation results. The parameters used in the experiments are presented in the Performance Evaluation section. The case studies can be accessed at https://infobiotics.org/casestudies/casestudies.html. The source files as well as the experimental data used in this paper are available to download at https://infobiotics.org/_static/experiments_2021.06.11.zip.

The Toggle Switch. The toggle switch is a seminal synthetic biological circuit that consists of two mutually repressing operons,[76] with *lacI* being regulated by a P*lambda* promoter and *cI* (a repressor of P*lambda*) being in turn regulated by P*lac*. The operon that codes for CI also codes for green fluorescent protein (GFP). Using this repressor pair allows the toggle switch to be activated by induction *via* IPTG and deactivated *via* aTc. The negative feedback cycle ensures that the switch maintain its on- or off-state even when the triggering inducer is washed out. The bistable behavior of the switch depends on the cooperativity of the repressor proteins—more specifically, on the right choice of regulators and promoters and their correct dimerization. Modeling with IBW can help develop a feasible design.

We simulate the toggle switch in four separate scenarios. We first suspended the cell in a region rich in IPTG and simulate its response in CI, LacI, and GFP production over a period of 1 h. We then use the average of the total CI, LacI, and GFP concentrations as initial values for CI, LacI, and GFP for simulation of the cell in an IPTG-depleted region. Again, the average final state after 1 h is used to initialize the cell state in a region rich in aTc, a.s.o. The averages and standard deviations of 50 simulation runs each are shown in Figure 4.

Verification is used to check the correctness of the system and to validate the system requirements using the probabilistic model checkers (see Table 1). The first property verifies that the suspension of the cell with IPTG will lead to the production of GFP. The second property verifies that once the cell is suspended with IPTG, the GFP concentration will eventually exceed 7.5 μM with a probability of 0.82. The third property verifies that GFP is produced at least three times more than CI despite the absence of IPTG with a probability 0.99. The fourth property verifies that the GFP production will decline if the cell is suspended with aTc.

To obtain a viable genetic sequence that implements the toggle switch, all that needs to be done is specify the sequences of the two promoters and three regulated proteins. We do this in IBW by pointing to the respective iGEM registry parts: BBa_R0010 for P*lac*, BBa_R0051 for P*lambda*, BBa_K105004 for *cI*, BBa_E0040 for *gfpmut3*, and BBa_C0012 for *lacI*. To reproduce a genetic arrangement of the published system, we introduce two compilation constraints that ensure that the second repressor operon is inverted (**ATGC DIRECTION: BACKWARD** in the respective device) and that the two promoters are adjacent to each other (**ATGC ARRANGE** Plambda,Plac).

Table 1. Verified Properties

#	verification statement	result
1	VERIFY [IPTG ≥ 10 μM] IS FOLLOWED BY [GFP > 7.5 μM]	T
2	VERIFY [IPTG ≥ 10 μM] IS FOLLOWED BY [GFP > 7.5 μM] WITH PROBABILITY ?	0.82
3	VERIFY [IPTG = 0 μM] IS FOLLOWED BY [GFP/CI > 3] WITH PROBABILITY ?	0.99
4	VERIFY [GFP] EVENTUALLY DECREASES WITH PROBABILITY ? GIVEN [aTc = 100 μM]	1.0

The ATGC biocompiler automatically fetches the sequence information, adds ribosome binding sites and terminators and assembles the concatenated gene sequence in accordance with the given layout constraints. The result of the biocompilation run is shown in Figure 5.

The Repressilator. An equally seminal early synthetic biological device is a synthetic genetic oscillator termed the "repressilator".[77] Similar to the previous toggle switch, the repressilator consists of mutually repressing repressors, but this time it features three instead of two chained interacting operons. As a result, none of the promoter activation states remains stable and the operons sequentially activate and inactivate each other—resulting in continued global oscillations.

Stochastic simulation confirms continued oscillations of the regulating proteins LacI, CI, and TetR (*cf.* Figure 6).

We can validate some system behavior using verification (see Table 2). Property 1 and 2 query the oscillatory behavior of LacI by checking the LacI concentration going below and above 1.25 μM repeatedly (with a probability 0.48 and 0.52, respectively). Property 3 verifies that the steady state concentration of CI is between 0.5 and 2 μM with probability 0.92.

The biocompiler output is shown in Figure 7.

Quorum Sensing. Quorum sensing is an ability expressed by certain bacterial species where a shift in population density leads to changes in gene expression. More specifically, *Pseudomonas aeruginosa* relies on a quorum sensing system comprised of two genes, *lasR* and *lasI*.[78] *lasR* codes for a transcriptional activator protein that is activated by 3-oxo-C12-HSL, an autoinducer synthesized by *lasI*. Their dimer promotes both *lasI* and *lasR* activity. At higher cell densities, the concentration of 3-oxo-C12-HSL rises, which leads to changes in *P. aeruginosa* associated with population density.

In the previous two case studies, we have shown how simulation and verification can be used together. The system dynamics can actually be analyzed using verification alone. In Table 3, we provide a list of queries that we can ask the model. For instance, one can verify the appearance of LasR later in the system, but not at the beginning, *i.e.*, the system will eventually lead to the production of the LasR protein (Property 1). The very same property can also be verified for 3-oxo-C12-HSL

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

Figure 5. Biocompiler result for the toggle switch specification. Sequence information of promoters, coding refions and terminators is drawn from several online repositories (here the iGem parts registry), and ribosome binding sites are automatically calculated using Salis' ribosome binding site calculator.

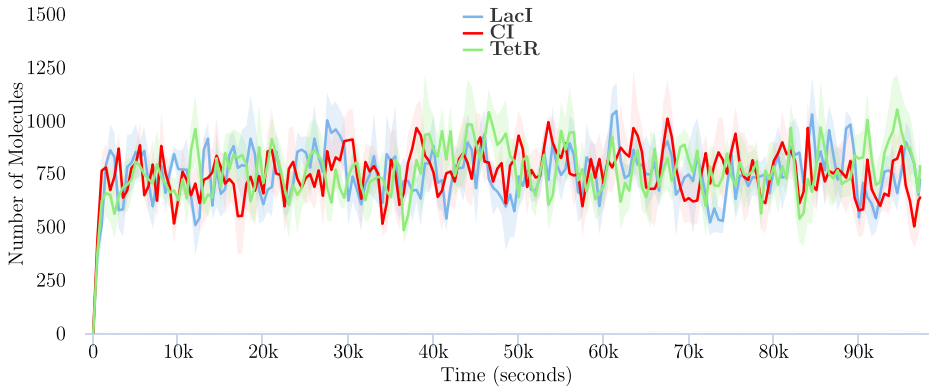


Figure 6. Simulation traces (mean and standard deviations of 50 runs using tau-leaping) for the LacI (blue), CI (red), and TetR (green) dimers over 27 h.

Table 2. Verified Properties

#	verification statement	result
1	VERIFY [LacI $\leq$ 1.25 μM] HOLDS INFINITELY OFTEN WITH PROBABILITY ?	0.48
2	VERIFY [LacI > 1.25 μM] HOLDS INFINITELY OFTEN WITH PROBABILITY ?	0.52
3	VERIFY [CI $\geq$ 0.5 μM] AND [CI < 2 μM]] HOLDS IN STEADY-STATE WITH PROBABILITY ?	0.92

(Property 2). This query can be complemented by a more grained analysis such as within a time period some threshold concentrations for LasR (Property 3) and 3-oxo-C12-HSL (Property 4) are reached. Finally, we verify it is very likely that the concentrations remain within an upper and lower bound in the steady-state (Property 5 and 6). From these results we can infer that the quorum sensing system works as described.

Genetic Logic Gates. In this section, we will demonstrate the compatibility of IBW with standard data exchange formats (*e.g.*, SBML and SBOL). IBW allows the users to upload SBML/SBOL files. Once such a model file is uploaded, the tool automatically translates the model to the IBL language. The users can then edit their model in IBL and add IBL specific annotations, *e.g.*, verification statements, part information, *etc.*, to utilize IBW features in their analysis.

Here, we will import some biological circuits based on logic gates implemented in SBML.[79] In our experiments, we consider the genetic NOT, AND, and NOR gates. All the models, originally in SBML, were automatically translated to IBL.

Each gate is composed of several genetic components, interacting with each other. For example, the AND gate circuit receives two proteins, LacI and TetR, as input. LacI and TetR inhibit two promoters that produce the CI protein. When the CI concentration goes below a certain threshold level, another promoter is activated, which produces green fluorescent protein (GFP) as output. The corresponding kinetic reactions are described by a number of equations captured in the SBML models. The circuit designs and their corresponding truth tables are shown in Figure 8.

Figure 9 shows the stochastic simulations of these gates. The simulation results show that the circuits exhibit the expected behavior. For example, for the NOT gate, the GFP concentration follows an opposite trend of the TetR concentration. For AND, the GFP concentration starts increasing as both LacI and TetR concentrations are high in the beginning; this is followed by a decrease in the GFP concentration as LacI and TetR concentrations decrease significantly. Similarly, for the NOR gate, the GFP

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

Figure 7. Biocompiler result for the repressilator specification.

Table 3. Verified Properties

#	verification statement	result
1	VERIFY [LasR] EVENTUALLY INCREASES WITH PROBABILITY ?	1.0
2	VERIFY [HSL] EVENTUALLY INCREASES WITH PROBABILITY ?	1.0
3	VERIFY [LasR $\geq$ 0.1 μM] EVENTUALLY HOLDS WITHIN [0,1] s WITH PROBABILITY ?	1.0
4	VERIFY [HSL$\geq$ 3 μM] EVENTUALLY HOLDS WITHIN [0,1] s WITH PROBABILITY ?	1.0
5	VERIFY [[LasR $\geq$ 0.005 μM] AND [LasR $\leq$ 0.15 μM]] HOLDS IN STEADY-STATE WITH PROBABILITY ?	0.98
6	VERIFY [[HSL $\geq$ 0.05 μM] AND [HSL $\leq$ 4.2 μM]] HOLDS IN STEADY-STATE WITH PROBABILITY ?	0.72

concentration remains very low as LacI and TetR concentrations are high; GFP is then observed in high concentration as both LacI and TetR degrade in time.

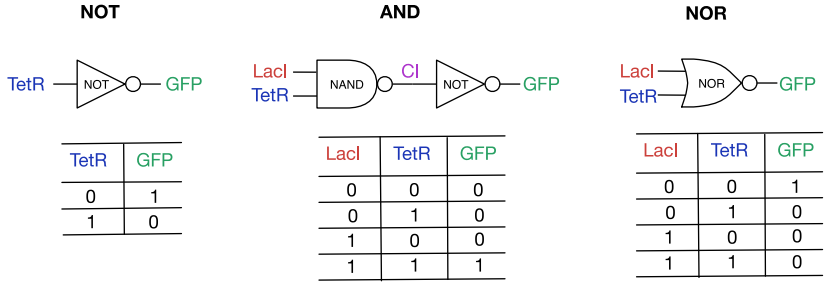
Although simulation results are useful to observe the general behaviors of the circuits, they fall short in determining the correct Boolean logic function. The challenge here is that the simulation data do not clearly show the classification of the *threshold* concentration levels into the Boolean values, "low" and "high".[80]

To address this limitation, we can utilize verification to make a more fine-grained analysis and hence can identify more accurate threshold values for these gates. The verification statements used in our experiments are presented in Table 4. Here, we assume the "high" input is achieved if the concentration of a species exceeds a threshold value (*Thr*), *e.g.*, GFP > *Thr*; similarly, the *low* input is triggered if the concentration level goes below the threshold, *e.g.*, GFP < *Thr*.

The Boolean logic function for each gate is captured by two verification properties (Table 4). For example, for the AND gate, the first property means that if the input proteins TetR and LacI are "high", the output protein GFP is also "high", whereas the second property means if any of the input proteins is "low", then the output protein is also "low".

The verification results are presented in Table 5. Here we assume that the highest probability result represents the most desired behavior. Since the Boolean logic function of each gate is defined by two properties, we consider the product of these probabilities as the "score" that represents the accuracy of a gate. The highest score defines the best threshold value. Using this method, we can also consider/identify different threshold values for different species.

We note that SBML only captures behavioral models of biological systems at the molecular level and it does not contain any genetic information on molecular species. Hence, there is not any genetic part information imported by IBW to run the biocompilation. SBOL models, on the other hand, describe structural and basic qualitative behavioral aspects with genetic part information, but the current SBOL standards do not include any quantitative information regarding molecular dynamics, hindering to run the simulation and verification.



TetR	GFP
0	1
1	0

LacI	TetR	GFP
0	0	0
0	1	0
1	0	0
1	1	1

LacI	TetR	GFP
0	0	1
0	1	0
1	0	0
1	1	0

Figure 8. Genetic NOT, AND, and NOR gate circuits, and the corresponding truth tables.

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

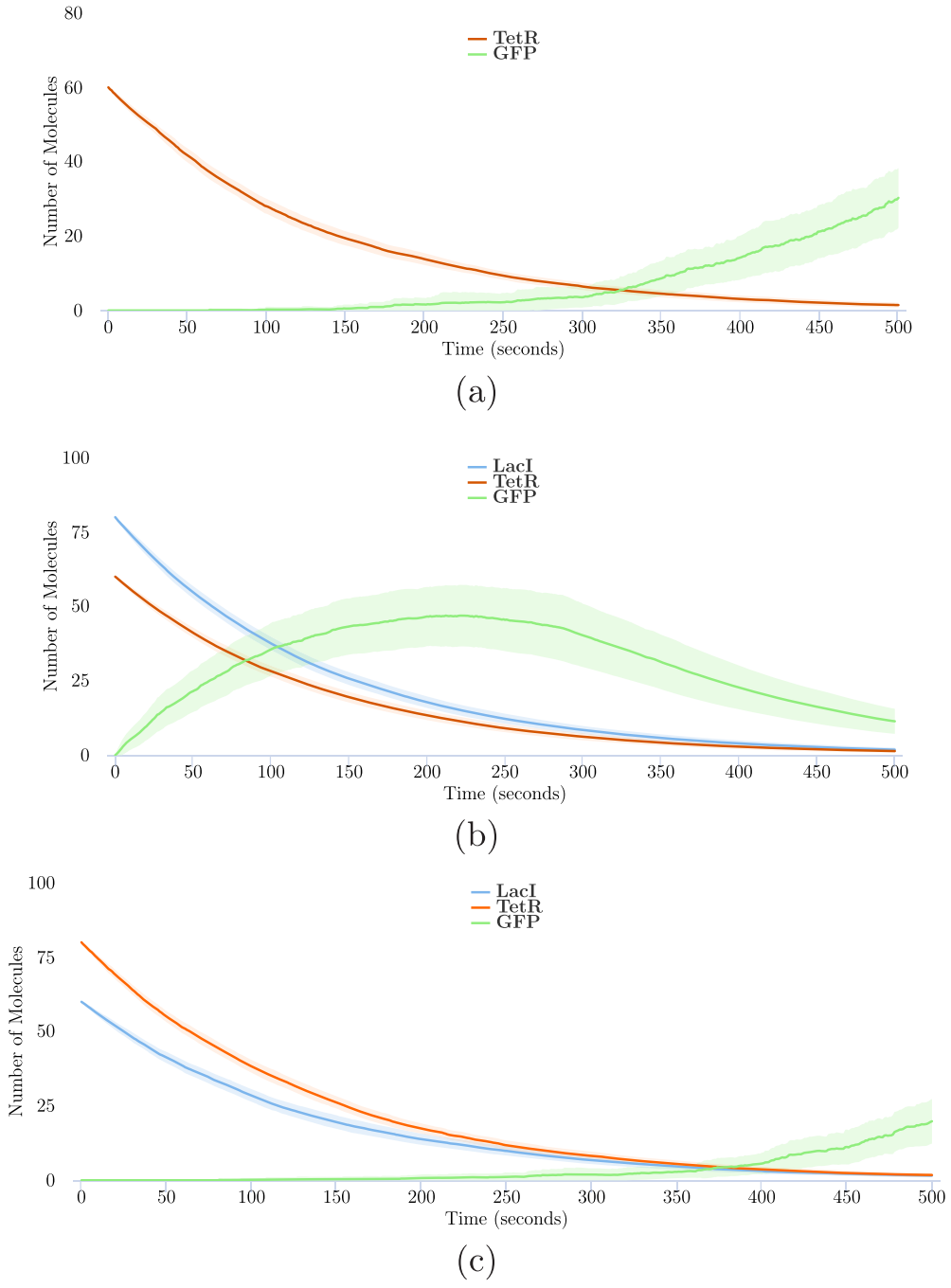


Figure 9. Simulation results. (a) NOT gate. (b) AND gate. (c) NOR gate.

Performance Evaluation. In this section, we provide a brief performance evaluation of the different steps. Table 6 presents the minimum and maximum times (in seconds) of each computational step for the case studies presented above.

The simulation experiments have been carried out using the GPU simulator based on the following VM configurations: 8 GB Memory, 6 Cores, Intel Core i7−4720HQ CPU @ 2.60 GHz. The verification experiments have been carried out using the statistical model checking which also utilizes the GPU simulator. As the results show, the simulation and verification run very fast thanks to the performance improvements provided the IBW. The biocompilation process for two case studies took around a minute. This is mainly due to the fact that all part sequences are obtained from online repositories (which requires setting up connection with remote servers) and the biocompilation process runs the RBS optimizer.

CONCLUSION

In this paper, we have presented the Infobiotics Workbench, a computer-aided design suite for synthetic biology, supporting an iterative workflow that begins with specification of the desired synthetic system and is followed by simulation and verification of the system in high-performance environments and ending with the eventual compilation of the system specification into a genetic construct.

IBW features a unique domain-specific language, providing a simple combined grammar for modeling, verification, and biocompilation statements. This novel approach offers

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

Table 4. Properties Verified for Each Gate[a]

circuit	#	verification statement
NOT	**1a**	VERIFY [TetR > *Thr* molecules] IS FOLLOWED BY [GFP < *Thr* molecules] WITH PROBABILITY ?
	1b	VERIFY [TetR < *Thr* molecules] IS FOLLOWED BY [GFP > *Thr* molecules] WITH PROBABILITY ?
AND	**2a**	VERIFY [[TetR > *Thr* molecules] AND [LacI > *Thr* molecules]] IS FOLLOWED BY [GFP > *Thr* molecules] WITH PROBABILITY ?
	2b	VERIFY [[TetR < *Thr* molecules] OR [LacI < *Thr* molecules]] IS FOLLOWED BY [GFP < *Thr* molecules] WITH PROBABILITY ?
NOR	**3a**	VERIFY [[TetR > *Thr* molecules] AND [LacI > *Thr* molecules]] IS FOLLOWED BY [GFP < *Thr* molecules] WITH PROBABILITY ?
	3b	VERIFY [[TetR < *Thr* molecules] OR [LacI < *Thr* molecules]] IS FOLLOWED BY [GFP > *Thr* molecules] WITH PROBABILITY ?

[a]*Thr* represents a threshold value.

Table 5. Verification Results with Respect to the Threshold (*Thr*) Value for Each Gate[a]

NOT			
Thr	$Prob_{1a}$	$Prob_{1b}$	score
5	0.69	0.95	0.66
10	**0.99**	**0.85**	**0.84**
15	1.0	0.77	0.77
20	1.0	0.65	0.65
25	1.0	0.55	0.55
30	1.0	0.50	0.50
AND			
Thr	$Prob_{2a}$	$Prob_{2b}$	score
5	1.0	0.18	0.18
10	1.0	0.47	0.47
15	0.98	0.76	0.74
20	0.98	0.90	0.88
25	**0.96**	**0.96**	**0.92**
30	0.95	0.97	0.92
NOR			
Thr	$Prob_{3a}$	$Prob_{3b}$	score
5	**0.96**	**0.82**	**0.79**
10	0.99	0.70	0.69
15	1.0	0.59	0.59
20	1.0	0.39	0.39
25	1.0	0.19	0.19
30	1.0	0.13	0.13

[a]Score is calculated by multiplying the probabilities obtained for two properties (*e.g.*, **1a** and **1b**).

Table 6. Simulation, Verification, and Biocompilation Times of the Case Studies[a]

case study	simulation (s)	verification (s)	biocompilation (s)
toggle switch	[0.11, 0.26]	[0.58, 0.91]	64
repressilator	[11.12, 13.56]	[12.9, 13.43]	59
quorum sensing	[12, 15]	[0.57, 0.68]	
logic gates	[0.10, 0.17]	[1.45, 4.5]	

[a]The experiments have been carried out using the following parameters: **toggle switch**: number of runs: 50, max time: 1000 s, interval: 10 s; **repressilator**: number of runs: 50, max time: 97 200 s (27 h), interval: 500 s; **quorum sensing**: number of runs: 50, max time: 10 ms, interval: 1 ms; **logic gates**: number of runs: 100, max time: 500 s, interval: 1 s.

seamless interoperability across different tools as well as compatibility with SBOL and SBML frameworks and removes the manual translation requirement for standalone applications.

We have shown the usability and applicability of the language and software using a number of case studies, *toggle switch*, *repressilator*, *quorum sensing*, and *logic gates*. For each case study, we have discussed the respective IBL specification, along with its verification, simulation, and biocompilation results.

IBW has quite unique features, but the current version has also some limitations. Currently, the language does not support multicellular bacterial populations. The tool can neither predict missing model parameters (*e.g.*, kinetic rates) nor optimize them. Our SBOL translator relies on the information provided in an SBOL file. If a part "role" is missing in the SBOL file, the translator cannot predict the role type, and assigns "UNKNOWN" role in the IBL translation, leaving this to the user to change manually. Also, the biocompiler only uses RBS optimizer; it does not do any other DNA optimization.

In our future work, we aim to extend the IBL language with a "spatial" dimension, so as to be able to represent specific geometries and strains. These will allow IBL to model specific formations, *e.g.*, biofilm coatings, and deal with large spatially distributed bacterial colonies. This extension entails investigating spatiotemporal logics and verification methods for synthetic biology. We will develop a web version of IBW to provide a more intuitive interface for experimental biologists. We will work on improving the RBS optimizer, which will reduce the biocompilation time. We will also broaden the spectrum of case studies. We will, in particular, explore more systems on synthetic drug design.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acssynbio.1c00143.

Table S1: a comparison of the features of the most well-known computer aided synthetic biology design tools (PDF)

■ AUTHOR INFORMATION

Corresponding Authors

Savas Konur − *Department of Computer Science, University of Bradford, Bradford BD7 1DP, U.K.*; orcid.org/0000-0002-0642-9452; Email: s.konur@bradford.ac.uk

Natalio Krasnogor − *Interdisciplinary Computing and Complex Biosystems Research Group, Newcastle University, Newcastle NE1 7RU, U.K.*; orcid.org/0000-0002-2651-4320; Email: natalio.krasnogor@newcastle.ac.uk

Authors

Laurentiu Mierla − *Department of Computer Science, University of Bradford, Bradford BD7 1DP, U.K.*

Harold Fellermann − *Interdisciplinary Computing and Complex Biosystems Research Group, Newcastle University, Newcastle NE1 7RU, U.K.*; orcid.org/0000-0001-5861-1945

Christophe Ladroue − *Department of Computer Science, University of Warwick, Coventry CV4 7AL, U.K.*

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

Bradley Brown − *Interdisciplinary Computing and Complex Biosystems Research Group, Newcastle University, Newcastle NE1 7RU, U.K.*
Anil Wipat − *Interdisciplinary Computing and Complex Biosystems Research Group, Newcastle University, Newcastle NE1 7RU, U.K.*; orcid.org/0000-0001-7310-4191
Jamie Twycross − *School of Computer Science, University of Nottingham, Nottingham NG8 1BB, U.K.*
Boyang Peter Dun − *Department of Computer Science, Stanford University, Stanford, California 94305, United States*
Sara Kalvala − *Department of Computer Science, University of Warwick, Coventry CV4 7AL, U.K.*
Marian Gheorghe − *Department of Computer Science, University of Bradford, Bradford BD7 1DP, U.K.*

Complete contact information is available at: https://pubs.acs.org/10.1021/acssynbio.1c00143

Author Contributions

S.Ko., L.M., H.F., S.Ka., M.G., and N.K. designed the research. S.Ko., L.M., H.F., and C.L. performed the research. S.Ko., L.M., H.F., C.L., B.B., J.T., and B.P.D. analyzed the data. S.Ko., H.F., and N.K. wrote the paper. L.M., C.L., B.B., A.W., J.T., B.P.D., S.Ka., and M.G. edited the paper.

Notes

The authors declare no competing financial interest.
The data and models that support the findings of this work are openly available at https://infobiotics.org/_static/experiments_2021.06.11.zip. All code associated with this manuscript is publicly available at https://github.com/Infobiotics/ibw.

■ ACKNOWLEDGMENTS

The work of S.K. is supported by EPSRC (EP/R043787/1). N.K., A.W., and B.B. acknowledge a Royal Academy of Engineering Chair in Emerging Technologies award and an EPSRC programme grant (EP/N031962/1).

■ REFERENCES

(1) Kuhn, P., Wagner, K., Heil, K., Liss, M., and Netuschil, N. (2017) Next generation gene synthesis: From microarrays to genomes. *Engineering in Life Sciences 17*, 6−13.
(2) Hsu, P. D., Lander, E. S., and Zhang, F. (2014) Development and applications of CRISPR-Cas9 for genome engineering. *Cell 157*, 1262−1278.
(3) Price, M., Cruz, R., Baxter, S., Escalettes, F., and Rosser, S. (2019) CRISPR-Cas9 In Situ engineering of subtilisin E in Bacillus subtilis. *PLoS One 14*, e0210121.
(4) Keating, K. W., and Young, E. M. (2019) Synthetic biology for bio-derived structural materials. *Curr. Opin. Chem. Eng. 24*, 107−114. Materials engineering: bioderived/bioinspired materials. Separations Engineering: advances in adsorption.
(5) Nguyen, P. Q. (2017) Synthetic biology engineering of biofilms as nanomaterials factories. *Biochem. Soc. Trans. 45*, 585−597.
(6) Gilbert, C., and Ellis, T. (2019) Biological Engineered Living Materials: Growing Functional Materials with Genetically Programmable Properties. *ACS Synth. Biol. 8*, 1−15.
(7) Lorenzo, V., Prather, K. L., Chen, G.-Q., O'Day, E., Kameke, C., Oyarzun, D. A, Hosta-Rigau, L., Alsafar, H., Cao, C., Ji, W., Okano, H., Roberts, R. J, Ronaghi, M., Yeung, K., Zhang, F., and Lee, S. Y. (2018) The power of synthetic biology for bioproduction, remediation and pollution control. *EMBO Rep. 19*, e45658.
(8) Gleizer, S., Ben-Nissan, R., Bar-On, Y. M., Antonovsky, N., Noor, E., Zohar, Y., Jona, G., Krieger, E., Shamshoum, M., Bar-Even, A., and Milo, R. (2019) Conversion of Escherichia coli to Generate All Biomass Carbon from CO_2. *Cell 179*, 1255−1263.
(9) Kylilis, N., Riangrungroj, P., Lai, H.-E., Salema, V., Fernández, L. A., Stan, G.-B. V., Freemont, P. S., and Polizzi, K. M. (2019) Whole-Cell Biosensor with Tunable Limit of Detection Enables Low-Cost Agglutination Assays for Medical Diagnostic Applications. *ACS Sensors 4*, 370−378.
(10) Sheth, R. U., and Wang, H. H. (2018) DNA-based memory devices for recording cellular events. *Nat. Rev. Genet. 19*, 718−732.
(11) Karr, J., Sanghvi, J., Macklin, D., Gutschow, M., Jacobs, J., Bolival, B., Assad-Garcia, N., Glass, J., and Covert, M. (2012) A Whole-Cell Computational Model Predicts Phenotype from Genotype. *Cell 150*, 389−401.
(12) Naylor, J., Fellermann, H., Ding, Y., Mohammed, W. K., Jakubovics, N. S., Mukherjee, J., Biggs, C. A., Wright, P. C., and Krasnogor, N. (2017) Simbiotics: A Multiscale Integrative Platform for 3D Modeling of Bacterial Populations. *ACS Synth. Biol. 6*, 1194−1210.
(13) Sanassy, D., Widera, P., and Krasnogor, N. (2015) Meta-Stochastic Simulation of Biochemical Models for Systems and Synthetic Biology. *ACS Synth. Biol. 4*, 39−47.
(14) Gorochowski, T. E., Hauert, S., Kreft, J.-U., Marucci, L., Stillman, N. R., Tang, T.-Y. D., Bandiera, L., Bartoli, V., Dixon, D. O. R., Fedorec, A. J. H., Fellermann, H., Fletcher, A. G., Foster, T., Giuggioli, L., Matyjaszkiewicz, A., McCormick, S., Montes Olivas, S., Naylor, J., Rubio Denniss, A., and Ward, D. (2020) Toward Engineering Biosystems With Emergent Collective Functions. *Front. Bioeng. Biotechnol. 8*, 705.
(15) Jiang, S., Wang, Y., Kaiser, M., and Krasnogor, N. (2020) NIHBA: a network interdiction approach for metabolic engineering design. *Bioinformatics 36*, 3482−3492.
(16) Misirli, G., Nguyen, T., McLaughlin, J. A., Vaidyanathan, P., Jones, T. S., Densmore, D., Myers, C., and Wipat, A. (2019) A Computational Workflow for the Automated Generation of Models of Genetic Designs. *ACS Synth. Biol. 8*, 1548−1559.
(17) Tellechea-Luzardo, J., Winterhalter, C., Widera, P., Kozyra, J., de Lorenzo, V., and Krasnogor, N. (2020) Linking Engineered Cells to Their Digital Twins: A Version Control System for Strain Engineering. *ACS Synth. Biol. 9*, 536−545.
(18) Watanabe, L., Nguyen, T., Zhang, M., Zundel, Z., Zhang, Z., Madsen, C., Roehner, N., and Myers, C. (2019) iBioSim 3: A Tool for Model-Based Genetic Circuit Design. *ACS Synth. Biol. 8*, 1560.
(19) Nielsen, A. A. K., Der, B. S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E. A., Ross, D., Densmore, D., and Voigt, C. A. (2016) Genetic circuit design automation. *Science 352*, aac7341.
(20) Gutiérrez, M., Gregorio-Godoy, P., Pérez del Pulgar, G., Muñoz, L. E., Sáez, S., and Rodríguez-Patón, A. (2017) A New Improved and Extended Version of the Multicell Bacterial Simulator GRO. *ACS Synth. Biol. 6*, 1496−1508.
(21) Chandran, D., and Sauro, H. M. (2012) Hierarchical modeling for synthetic biology. *ACS Synth. Biol. 1*, 353−364.
(22) Bilitchenko, L., Liu, A., Cheung, S., Weeding, E., Xia, B., Leguia, M., Anderson, J. C., and Densmore, D. (2011) EUGENE - A domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS One 6*, e18882.
(23) Beal, J., Lu, T., and Weiss, R. (2011) Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks. *PLoS One 6*, e22490.
(24) Villalobos, A., Ness, J. E., Gustafsson, C., Minshull, J., and Govindarajan, S. (2006) Gene Designer: a synthetic biology tool for constructing artificial DNA segments. *BMC Bioinf. 7*, 285.
(25) Gaspar, P., Oliveira, J. L., Frommlet, J., Santos, M., and Moura, G. (2012) EuGene: maximizing synthetic gene design for heterologous expression. *Bioinformatics 28*, 2683.
(26) Czar, M. J., Cai, Y., and Peccoud, J. (2009) Writing DNA with GenoCAD. *Nucleic Acids Res. 37*, W40−W47.
(27) Pedersen, M., and Phillips, A. (2009) Towards programming languages for genetic engineering of living cells. *J. R. Soc., Interface 6*, S437−S450.

(28) Misirli, G., Hallinan, J. S., Yu, T., Lawson, J. R., Wimalaratne, S. M., Cooling, M. T., and Wipat, A. (2011) Model annotation for synthetic biology: automating model to nucleotide sequence conversion. *Bioinformatics 27*, 973−979.
(29) Swainston, N., Dunstan, M., Jervis, A. J., Robinson, C. J., Carbonell, P., Williams, A. R., Faulon, J.-L., Scrutton, N. S., and Kell, D. B. (2018) PartsGenie: an integrated tool for optimizing and sharing synthetic biology parts. *Bioinformatics 34*, 2327−2329.
(30) Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. (2006) COPASI—a COmplex PAthway SImulator. *Bioinformatics 22*, 3067.
(31) Moraru, I., Schaff, J., and Slepchenko, B. (2008) Virtual cell modelling and simulation software environment. *IET Syst. Biol. 2*, 352.
(32) Galdzicki, M., Clancy, K. P., Oberortner, E., Pocock, M., Quinn, J. Y., Rodriguez, C. A., Nicholas, R., Wilson, M. L., Adam, L., Anderson, J. C., et al. (2014) The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat. Biotechnol. 32*, 545−550.
(33) Hucka, M., et al. (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics 19*, 524.
(34) Oberortner, E., Densmore, D., and Anderson, J. C. (2012) An interactive pattern story on designing the architecture of Clotho. In *Proceedings of the 19th Conference on Pattern Languages of Programs*, Association for Computing Machinery.
(35) Blakes, J., Twycross, J., Romero-Campero, F. J., and Krasnogor, N. (2011) The Infobiotics Workbench: an integrated *in silico* modelling platform for Systems and Synthetic Biology. *Bioinformatics 27*, 3323−3324.
(36) Mısırlı, G., Taylor, R., Goñi-Moreno, A., McLaughlin, J. A., Myers, C., Gennari, J. H., Lord, P., and Wipat, A. (2019) SBOL-OWL: An Ontological Approach for Formal and Semantic Representation of Synthetic Biology Information. *ACS Synth. Biol. 8*, 1498−1514.
(37) SyBiOnt: The Synthetic Biology Ontology: http://sybiont.org.
(38) Gene Ontology: http://geneontology.org.
(39) SequenceOntology: http://www.sequenceontology.org.
(40) Sanassy, D., Fellermann, H., Krasnogor, N., Konur, S., Mierla, L. M., Gheorghe, M., Ladroue, C., and Kalvala, S. (2014) Modelling and stochastic simulation of synthetic biological boolean gates. In *High Performance Computing and Communications, 2014 IEEE 6th Intl. Symp. on Cyberspace Safety and Security, 2014 IEEE 11th Intl. Conf. on Embedded Software and Syst. (HPCC, CSS, ICESS)*, pp 404−408, IEEE.
(41) EMBL-EBI: https://www.ebi.ac.uk.
(42) Hillis, W. D., and Steele, G. L. (1986) Data Parallel Algorithms. *Commun. ACM 29*, 1170−1183.
(43) Konur, S. (2006) A Decidable Temporal Logic for Events and States. *In Thirteenth International Symposium on Temporal Representation and Reasoning (TIME'06)*, pp 36−41.
(44) Konur, S. (2008) An Interval Logic for Natural Language Semantics. In *Proceedings of the Seventh conference on Advances in Modal Logic*, Nancy, France, pp 177−191.
(45) Konur, S. (2014) Specifying Safety-critical Systems with a Decidable Duration Logic. *Science of Computer Programming 80*, 264−287.
(46) Alur, R., McMillan, K., and Peled, D. (2000) Model-checking of correctness conditions for concurrent objects. *Information and Computation 160*, 167−188.
(47) Yabandeh, M. (2011) *Model checking of distributed algorithm implementations*. Ph.D. thesis, École Polytechnique Fédérale de Lausanne.
(48) Konur, S., and Fisher, M. (2011) Formal Analysis of a VANET Congestion Control Protocol through Probabilistic Verification. In *Proceedings of the 73rd IEEE Vehicular Technology Conference, VTC Spring 2011*, May 15−18, 2011, Budapest, Hungary, pp 1−5.
(49) Konur, S. (2014) Towards Light-Weight Probabilistic Model Checking. *J. Appl. Math. 2014*, 15.
(50) Abbink, H., van Dijk, R., Dobos, T., Hoogendoorn, M., Jonker, C., Konur, S., van Maanen, P.-P., Popova, V., Sharpanskykh, A., van Tooren, P., Treur, J., Valk, J., Xu, L., and Yolum, P. (2004) Automated Support for Adaptive Incident Management. In *Proceedings of ISCRAM'04, Brussels*, pp 153−170.
(51) Konur, S., Fisher, M., and Schewe, S. (2013) Combined model checking for temporal, probabilistic, and real-time logics. *Theoretical Computer Science 503*, 61−88.
(52) Arapinis, M., Calder, M., Denis, L., Fisher, M., Gray, P., Konur, S., Miller, A., Ritter, E., Ryan, M., Schewe, S., Unsworth, C., and Yasmin, R. (2009) Towards the verification of pervasive systems. *Electron. Commun. EASST*,, http://eprints.gla.ac.uk/39287/.
(53) Konur, S., Fisher, M., Dobson, S., and Knox, S. (2014) Formal Verification of a Pervasive Messaging System. *Formal Aspects of Computing 26*, 677−694.
(54) Konur, S., and Fisher, M. (2015) A roadmap to pervasive systems verification. *Knowledge Engineering Review 30*, 324−341.
(55) Konur, S., Dixon, C., and Fisher, M. (2010) *Formal Verification of Probabilistic Swarm Behaviours*; pp 440−447, Swarm Intelligence, Berlin, Heidelberg.
(56) Konur, S., and Gheorghe, M. (2015) A Property-Driven Methodology for Formal Analysis of Synthetic Biology Systems. In *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp 360−371.
(57) Camci, F., Eker, O. F., Baskan, S., and Konur, S. (2016) Comparison of sensors and methodologies for effective prognostics on railway turnout systems. *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit 230*, 24−42.
(58) Lefticaru, R., Konur, S., Yildirim, Ü., Uddin, A., Campean, F., and Gheorghe, M. (2017) Towards an Integrated Approach to Verification and Model-Based Testing in System Engineering. In *The International Workshop on Engineering Data- & Model-driven Applications (EDMA-2017)*, pp 131−138.
(59) Lefticaru, R., Bakir, M. E., Konur, S., Stannett, M., and Ipate, F. (2018) Modelling and Validating an Engineering Application in Kernel P Systems. In *Membrane Computing*, pp 183−195.
(60) Konur, S. (2010) A Survey on Temporal Logics. *arXiv*, May 18, 2010, arXiv:1005.3199
(61) Konur, S. (2010) Real-time and Probabilistic Temporal Logics: An Overview. *arXiv*, May 18, 2010, arXiv:1005.3200
(62) Konur, S. (2011) An Event-Based Fragment of First-Order Logic over Intervals. *Journal of Logic, Language and Information 20*, 49−68.
(63) Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. (1999) Patterns in Property Specifications for Finite-state Verification. In *Proceedings of the 21st International Conference on Software Engineering*, pp 411−420.
(64) Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M. (1999) NuSMV: A new symbolic model verifier. In *International Conference on Computer Aided Verification*, pp 495−499.
(65) Kwiatkowska, M., Norman, G., and Parker, D. (2002) PRISM: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pp 200−204.
(66) Grosu, R., and Smolka, S. A. (2005) Monte Carlo model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp 271−286.
(67) Baier, C., and Katoen, J.-P. (2008) *Principles of Model Checking (Representation and Mind Series)*, The MIT Press.
(68) Sen, K., Viswanathan, M., and Agha, G. (2004) *Computer Aided Verification*, Lecture Notes in Computer Science, Vol. *3114*; pp 202−215, Springer, Berlin.
(69) Legay, A., Delahaye, B., and Bensalem, S. (2010) *Runtime Verification*, Lecture Notes in Computer Science, Vol. *6418*; pp 122−135, Springer, Berlin.
(70) Ladroue, C., and Kalvala, S. (2015) Constraint-Based Genetic Compilation. In *International Conference on Algorithms for Computational Biology*, pp 25−38.

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945

(71) Baker, D., Church, G., Collins, J., Endy, D., Jacobson, J., Keasling, J., Modrich, P., Smolke, C., and Weiss, R. (2006) Engineering life: building a fab for biology. *Sci. Am. 294*, 44−51.
(72) Roberts, R. J., Vincze, T., Posfai, J., and Macelis, D. (2004) REBASE-restriction enzymes and DNA methyltransferases. *Nucleic acids research 33*, D230−D232.
(73) Salis, H. M., Mirsky, E. A., and Voigt, C. A. (2009) Automated design of synthetic ribosome binding sites to control protein expression. *Nat. Biotechnol. 27*, 946−950.
(74) Hillson, N. J., Rosengarten, R. D., and Keasling, J. D. (2012) j5 DNA assembly design automation software. *ACS Synth. Biol. 1*, 14−21.
(75) Crowther, M., Grozinger, L., Pocock, M., Taylor, C. P. D., McLaughlin, J. A., Mısırlı, G., Bartley, B. A., Beal, J., Goñi-Moreno, A., and Wipat, A. (2020) ShortBOL: A Language for Scripting Designs for Engineered Biological Systems Using Synthetic Biology Open Language (SBOL). *ACS Synth. Biol. 9*, 962−966.
(76) Gardner, T. S., Cantor, C. R., and Collins, J. J. (2000) Construction of a genetic toggle switch in Escherichia coli. *Nature 403*, 339−342.
(77) Elowitz, M. B., and Leibler, S. (2000) A synthetic oscillatory network of transcriptional regulators. *Nature 403*, 335−338.
(78) Dockery, J. D., and Keener, J. P. (2001) A Mathematical Model for Quorum Sensing in Pseudomonas aeruginosa. *Bull. Math. Biol. 63*, 95.
(79) Myers, C. (2010) *Engineering Genetic Circuits*, CRC Press.
(80) Baig, H., and Madsen, J. (2017) Simulation Approach for Timing Analysis of Genetic Logic Circuits. *ACS Synth. Biol. 6*, 1169−1179.

https://doi.org/10.1021/acssynbio.1c00143
ACS Synth. Biol. 2021, 10, 1931−1945