Catalyst: Fast Biochemical Modeling with Julia

Torkel E. Loman*† Yingbo Ma ‡ Vasily Ilin§ Shashi Gowda ¶ Niklas Korsbo ‖
Nikhil Yewale** Chris Rackauckas*†† Samuel A. Isaacson*‡‡

July 30, 2022

Abstract

We introduce Catalyst.jl, a flexible and feature-filled Julia library for modeling and high performance simulation of chemical reaction networks (CRNs). Catalyst acts as both a domain-specific language and an intermediate representation for symbolically encoding CRN models as Julia-native objects. This enables a pipeline of symbolically specifying, analyzing, and modifying reaction networks; converting Catalyst models to symbolic representations of concrete mathematical models; and generating compiled code for use in numerical solvers. Currently Catalyst supports conversion to symbolic discrete stochastic chemical kinetics (jump process), chemical Langevin (stochastic differential equation), and mass-action reaction rate equation (ordinary differential equation) models. Leveraging ModelingToolkit.jl and Symbolics.jl, Catalyst models can be analyzed, simplified, and compiled into optimized representations for use in a broad variety of numerical solvers. The performance of the numerical solvers Catalyst targets is illustrated across a variety of reaction networks by benchmarking stochastic simulation algorithm and ODE solver performance. We demonstrate the extendability and composability of Catalyst by highlighting both how it can compose with a variety of Julia libraries, and how existing open source projects have extended the intermediate representation. These benchmarks demonstrate significant performance improvements compared to several popular reaction network simulators.

Introduction

Chemical reaction network (CRN) models are used across a variety of fields, including the biological sciences, epidemiology, physical chemistry, combustion modeling, and pharmacology. At their core, they combine a set of species (defining a system's state) with a set of reaction events (rates for reactions occuring, with rules for altering the system's state when a reaction occurs). One advantage with the formulation of a biological model as a CRN is that these can be simulated according to several well-defined mathematical representations, representing different physical scales at which reaction processes can be studied. For example, the reaction rate equation (RRE) is a macroscopic system of ordinary differential equations (ODEs), providing a deterministic model of chemical reaction processes. Similarly, the chemical Langevin equation (CLE) is a system of stochastic differential equations (SDEs), providing a more microscopic model that can capture certain types of fluctuations in reaction processes [16]. Finally, stochastic chemical kinetics, typically simulated with Gillespie's algorithm (as well as modifications to, and improvements of, it), provides an even more microscopic model, that captures both stochasticity and discreteness of populations in chemical reaction processes [14, 15]. That a CRN can be unambiguously represented using these models forms the basis of several CRN modeling tools [36, 23, 18, 26, 41, 27, 38, 21]. Here we present a new modeling tool for CRNs, Catalyst.jl, which we believe offers a unique set of advantages for both inexperienced and experienced modelers.

Catalyst's defining trait, which sets it apart from other popular CRN modeling packages, is that it represents models in an entirely symbolic manner, accessible via standard Julia programs. This permits algebraic manipulation and simplification of the models, either by the user, or by other tools. Once a CRN has been defined, it is stored in a symbolic *intermediate representation* (IR). This IR is the target of methods that provide functionality to Catalyst, including numerical solvers for both continuous ODEs and SDEs, as well as discrete Gillespie-style stochastic simulation algorithms (SSAs). As Catalyst's symbolic representations can be converted to compiled Julia functions, it can be easily used with a variety of Julia libraries. These include packages for parameter fitting, sensitivity analysis, steady state finding, and bifurcation analysis. Finally, to simplify model implementation, Catalyst provides a *domain-specific language* (DSL) that allows users to declare CRN models

*These authors contributed equally.
†Sainsbury Laboratory, University of Cambridge
‡Julia Computing
§Department of Mathematics, University of Washington, vilin@uw.edu
¶Department of Computer Science, Massachusetts Institute of Technology
‖Pumas-AI
**Department of Applied Mechanics, IIT Madras, Chennai-600 036, India
††Computer Science and AI Laboratory (CSAIL), Massachusetts Institute of Technology, Julia Computing, Pumas-AI, crackauc@mit.edu
‡‡Department of Mathematics and Statistics, Boston University, isaacsas@bu.edu

using classic chemical reaction notation.

Catalyst is implemented in Julia, a new (version 1.0 released in August 2018) open-source programming language for scientific computing. Its combination of high performance and user-friendliness makes it highly promising [4, 3]. It has grown quickly, and already has a highly developed ecosystem of available packages for scientific simulation. This includes the Scientific Machine Learning (SciML) organization, of which Catalyst is a part. SciML, through its ModelingToolkit.jl package, provides the IR on which Catalyst is based [28]. This IR is used across the organization's projects, providing a target structure both for model-generation tools (such as Catalyst), and tools that provide additional functionality. ModelingToolkit symbolic models leverage the Symbolics.jl [19] *computer algebraic system* (CAS), enabling them to be represented in a symbolic manner. Simulations of ModelingToolkit-based models are typically carried out using DifferentialEquations.jl, perhaps the largest software package of state-of-the-art, high performance numerical solvers for ODEs, SDEs and jump processes [35].

Several existing modeling packages provide overlapping functionality with Catalyst. COPASI is a well known and popular software that enables both deterministic and stochastic CRN modeling, as well as many auxiliary features (such as parameter fitting and sensitivity analysis) [23]. BioNetGen is another such software suite, currently available as a Visual Studio Code extension, that is built around the popular BioNetGen language for easily specifying complex reaction network models [21]. It provides options for model creation, network simulation, and network free-modeling. Another popular tool, VCell provides extensive functionality, via an intuitive graphical interface [36]. Other modeling tools such as GINsim, CellNOpt, gillespy2, and Matlab's SimBiology are more limited in their target scope [18, 1, 41].

Compared to these packages, Catalyst has immediate access to a more extensive set of numerical solvers for ODEs, SDEs (which COPASI lacks), and SSAs. It also has the ability to include Julia-native functions within rate laws and stoichiometric expressions, and to include coupled ODEs or algebraic constraint equations (potentially resulting in differential-algebraic equations (DAEs)). For example, to encode bursty reactions stoichiometric coefficients can be defined using standard Julia functions that sample from a random variable distribution. Similarly, rate-laws can include data-driven modeling terms constructed via Julia libraries such as Surrogates.jl, SciMLSensitivity.jl, and DiffEqFlux.jl. Moreover, Catalyst generates differentiable models, which can be easily incorporated into higher-level Julia codes and composed with other Julia libraries.

In contrast to several other packages, Catalyst does not currently provide easy generation of hybrid methods that allow model components to be defined at different physical scales (such as resolving some reactions via ODEs and others via jump processes). Catalyst is also DSL and API-based, with simulation and analysis of models carried out via Julia programs as opposed to the GUIs of BioNetGen, COPASI, and VCell.

In the next sections we overview a basic workflow for using Catalyst to define and simulate CRNs; overview how Catalyst performs relative to several popular CRN modeling packages for solving ODEs and simulating jump processes; discuss Catalyst's symbolic representation of CRNs, Catalyst's network analysis functionality, and how it can compose with other Julia packages; and introduce some of the higher-level applications in which Catalyst models can be easily embedded.

Results

The Catalyst DSL enables models to be created using chemical reaction notation

Catalyst offers several ways to define a CRN model, with the most effortless option being the `@reaction_network` DSL. This feature extends the natural Julia syntax via a macro, allowing users to declare CRN models using classic chemical reaction notation (as opposed to declaring models using equations, or by declaring reactions implicitly or through functions). This alternative notation makes scripts more human legible, and greatly reduces code length (simplifying both script writing and debugging). Using the DSL, the CRN's chemical reactions are listed, each preceded by its reaction rate (Figure 1). From this, the system's species are automatically extracted and a `ReactionSystem` IR structure is created (which can be used as input to e.g. numerical simulators).

To facilitate a more concise notation, similar reactions (e.g. several degradation events) can be bundled together. Reaction rates can either be a constant, a parameter, or a function (such as a Hill function). Both non-integer and parametric stoichiometric coefficients are possible. There are also several non-DSL methods for model creation. They include loading networks from files via SBMLToolkit.jl (for SBML files) and ReactionNetworkImporters.jl (for BioNetGen generated .net files). CRNs can also be created via defining symbolic variables in ModelingToolkit, and directly building `ReactionSystem`s from collections of `Reaction` structures. This enables programmatic definition of CRNs, making it possible to create large models by iterating through a relatively small number of rules using standard Julia language scripts.

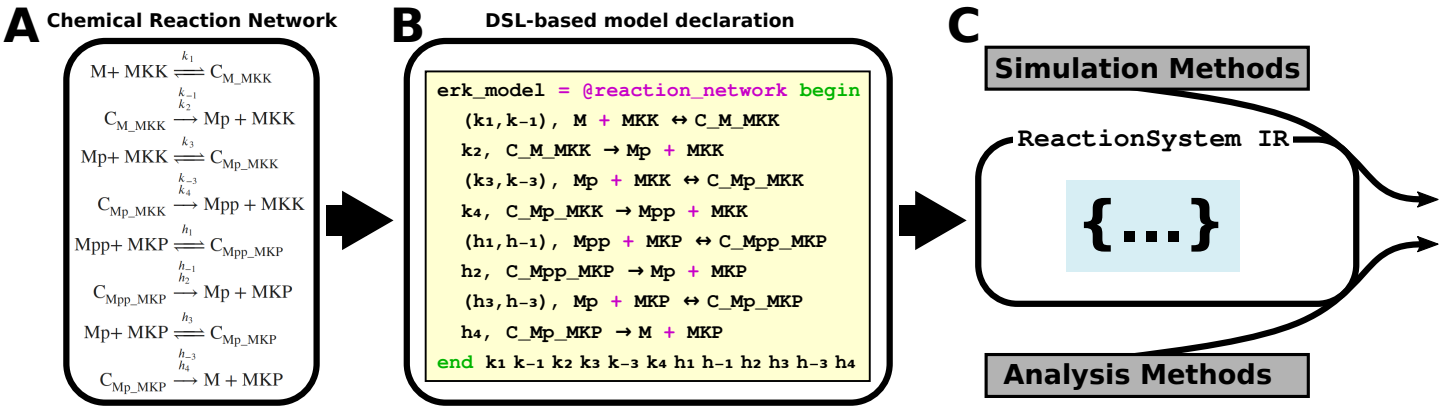

Figure 1: **Catalyst connects an intuitive domain-specific language with a well supported intermediate representation.** The extracellular signal-regulated kinase (ERK) network is important to the regulation of many cellular functions, and its disruption has been implicated in cancer [31]. (A) Here, a CRN representation of the ERK network is shown. (B) A model of the ERK CRN can be implemented in Julia through the Catalyst DSL, using code very similar to the actual CRN representation. (C) From this code, the DSL generates a `ReactionSystem` IR that is the target structure for a range of supported simulation and analysis methods.

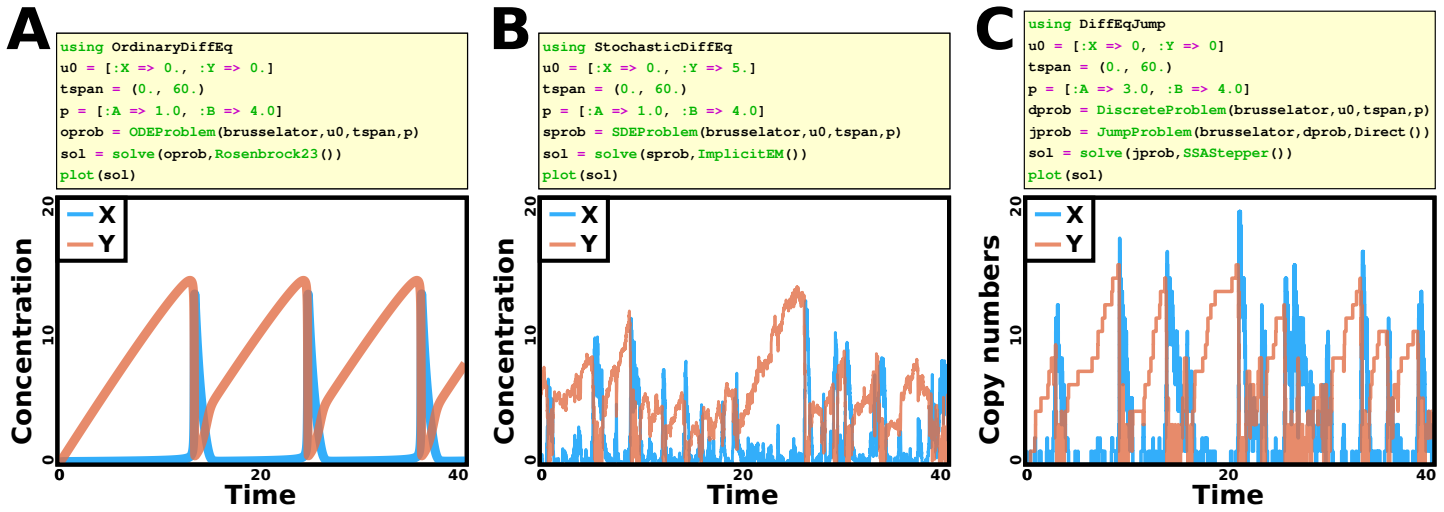

Figure 2: **Catalyst models can be simulated using both deterministic and stochastic interpretations.** The well known brusselator CRN contains two species (X and Y) and depends on two parameters (A and B). Here, it is simulated using three different interpretations, in each case both the code to make and plot the simulation, as well as the plot is shown. Note, the code to define the `brusselator ReactionSystem` model is not shown, but is analogous to the DSL code for the `erk_model` in Figure 1. Additional post-processing has been carried out on the plot to improve their visualization in the article format. (A) While $B > 1 + A^2$, the deterministic interpretation of the model exhibits a limit cycle. This is confirmed using RRE simulations. (B) The model can also be simulated using the stochastic CLE interpretation. (C) Finally, the discrete, stochastic, jump process interpretation is simulated via Gillespie's direct method. Here, the system displays a limit cycle even though $B < 1 + A^2$, confirming the well known phenomenon of noise induced oscillations [46].

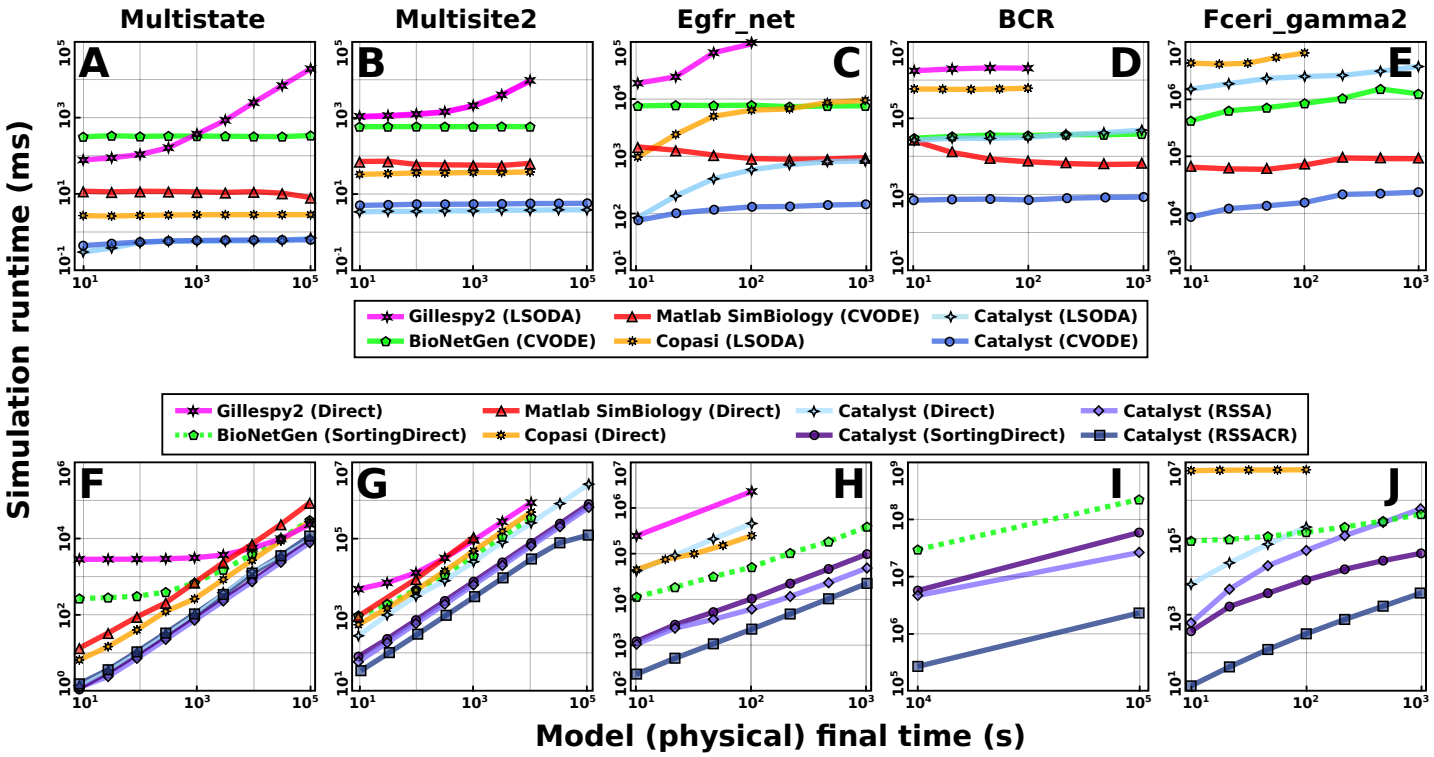

Figure 3: **Simulations of Catalyst models outperform those of other modeling packages.** Benchmarking of simulation runtimes against the final (physical) time at which models are simulated to for Catalyst and four other modeling packages (gillespy2, BioNetGen, Matlab SimBiology, and COPASI). The benchmarks were run on the multi-state (Multistate, 9 species and 18 reactions [39]), multi-site (Multisite 2, 66 species and 288 reactions [9]), epidermal growth factor receptor signalling (Egfr_net, 356 species and 3749 reactions [5]), B-cell receptor (1122 species and 24388 reactions [2]), and high-affinity human IgE receptor signalling (Fceri_gamma2, 3744 species and 58276 reactions [11]) models. Catalyst typically outperforms the other packages by one to two orders of magnitude. (A-E) Benchmarking deterministic RRE ODE simulations of the five models. Identical values for absolute and relative tolerance parameters are used for all packages. (F-J) Benchmarking of stochastic chemical kinetics SSA simulations of the five models. Via JumpProcesses.jl, Catalyst can use several different algorithms for exact Gillespie simulations, a subset of which are shown. Note, it was remarked in [20] that BioNetGen (dotted green lines) use a pseudo-random number generator in SSAs that, while fast, is lower quality than many (slower) modern generators such as Mersenne Twister.

Catalyst models can be simulated using a wide range of high-performance methods

Numerical simulations of Catalyst models are generally carried out using the DifferentialEquations package. It contains a large number of numerical solvers and a wide range of additional features (such as event handling and automatic parallelization). The package is highly competitive, often outperforming packages written in C and Fortran [35]. Simulation syntax is straightforward, and output solutions can be plotted using the Plots.jl package [7] via a recipe that allows users to select the species and times to display. CRNs can be translated and simulated using three interpretations, the ODE based RRE, the SDE based CLE, and through discrete SSAs (Figure 2).

To demonstrate the performance of these solvers, we benchmarked simulations of CRN models using a range of CRN modeling tools (BioNetGen, Catalyst, COPASI, gillespy2, and Matlab's SimBiology toolbox) (Methods). We used both ODE simulations and discrete SSAs. Few packages permit SDE simulations, hence such simulations were not benchmarked. Note, however, DifferentialEquations' SDE solvers are highly performative [34]. When comparing a range of models, from small to large, we see that Catalyst typically outperforms the other packages, often by one to two orders of magnitude or more (Figure 3). Especially for Gillespie-style simulations, the wide range of methods provided by DifferentialEquations enables Catalyst to outperform the other packages (most of which only uses Gillespie's direct method).

Catalyst enables composable, symbolic modeling of CRNs

Catalyst's primary feature is that its models are represented using a CAS, enabling them to be algebraically manipulated. Examples of how this is utilised include automatic computation of system Jacobians, calculation and elimination of conservation laws, and simplification of generated symbolic DAE models via ModelingToolkit's symbolic analysis tooling. These techniques can help speed up numeric simulations, while also facilitating higher level analysis (for example, by generating

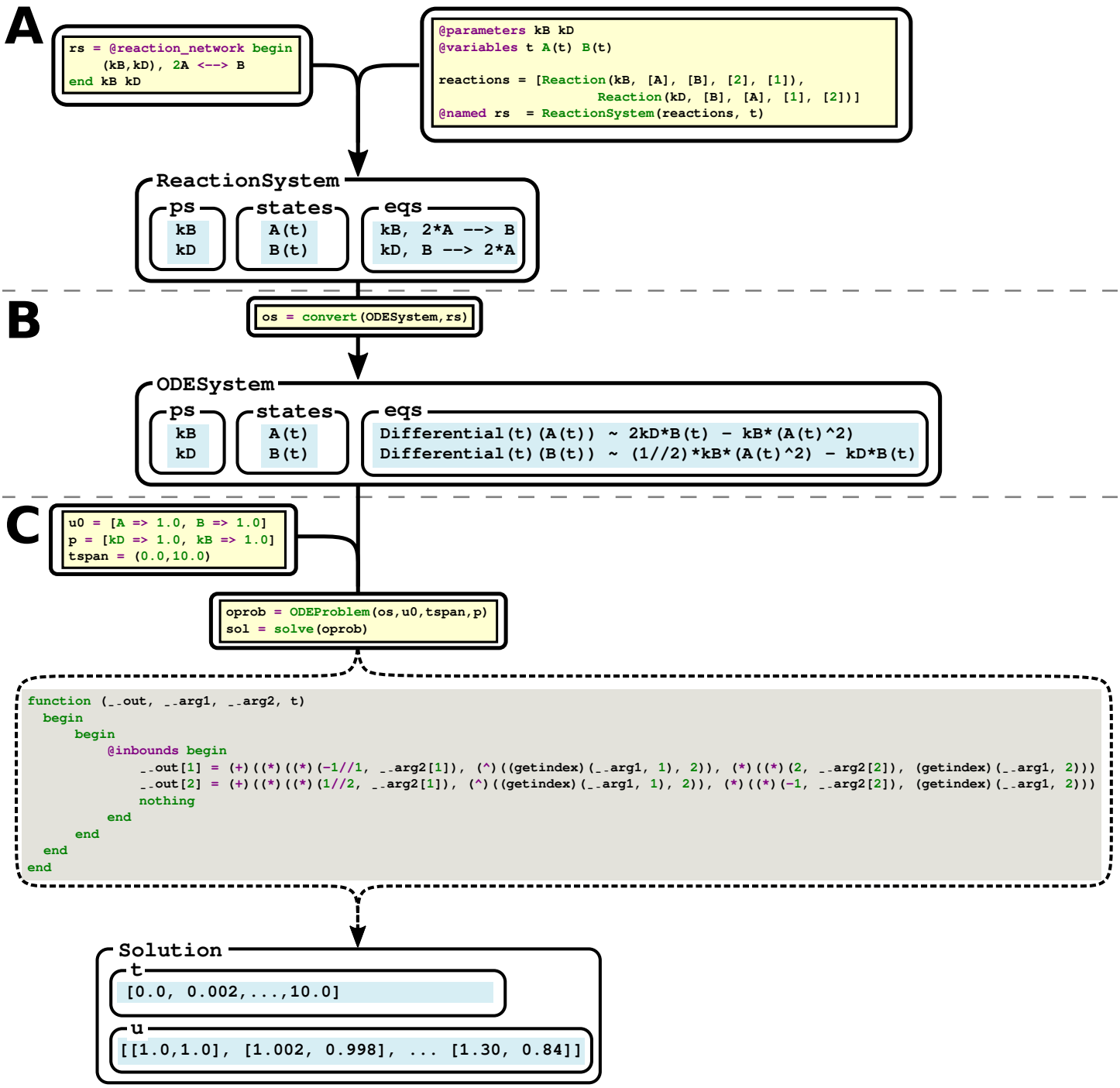

Figure 4: **The simulation of a Catalyst model, with internal intermediates displayed.** Code as written by the user (yellow background), and as generated internally by Catalyst and ModelingToolkit (blue and grey backgrounds respectively) are shown, in addition to the generated structures and their fields (some of the internal fields are omitted in all displayed structures). (A) A reaction system (consisting of a reversible dimerisation reaction) is created using either the DSL, or programmatically using the CAS. The model is stored in a `ReactionSystem` structure. (B) The `ReactionSystem` can be converted into a ModelingToolkit `ODESystem` structure, corresponding to a symbolic RRE ODE model. (C) By providing initial conditions, parameter values, and a time span, the `ODESystem` can be simulated, generating an output solution. The generated (internal) Julia code for evaluating the derivatives defining the ODEs, which gets compiled and is input to the ODE solver, is displayed in grey. At each step, the user has the ability to investigate and manipulate the generated structures.

non-singular Jacobians after elimination of conservation laws that can aid steady-state analysis). The symbolic representation also permits model internals to be freely extracted, investigated, and manipulated, giving the user full control over their models (Figure 4). This enables various forms of programmatic model creation, extension and composition. Model structures that occur repetitively can be duplicated, and disjoint models can be connected together. For example, such functionality can be used to model a population of cells, each with defined neighbours, where each cell can be assigned a duplicate of the same simple CRN. The CRNs within each cell can then be connected to those of its neighbours, enabling models with transport structures. Similarly, one could define a collection of genetic modules, and then compose such modules together into a larger gene regulatory network.

Catalyst is highly flexible in the allowed Julia functions that can be used in defining rates, rate laws, or stoichiometry coefficients. This means that while reaction rates and rate laws are typically constants, parameters, or simple functions, e.g. Hill equations, they may also include other terms, such as neural networks or data-driven, empirically defined, Julia functions. Likewise, stoichiometric coefficients can be random variables by defining them as a symbolic variable, and setting that variable equal to a Julia function sampling the appropriate probability distribution. Such functionality can be utilized, for example, to model transcriptional bursting [17], where the produced mRNA copy-numbers are random variables. Finally, standard Catalyst-generated ODE and SDE models are differentiable, in that the generated codes can be used in higher-level packages that rely on automatic differentiation. In this way Catalyst-generated models can be used in machine-learning based analyses.

That Catalyst gives full access to its model internals, combined with its composability, allows other packages to easily integrate into, and build upon, it. Indeed, this is already being utilised by independent package developers. The MomentClosure.jl Julia package, which implements several techniques for moment closure approximations, is built to be deployed on Catalyst models [40]. It can generate symbolic finite-dimensional ODE system approximations to the full, infinite system of moment equations associated with the chemical master equation. These symbolic approximations can then be compiled and solved via ModelingToolkit in a similar manner to how Catalyst's generated RRE ODE models are handled. Similarly, FiniteStateProjection.jl [32] builds upon Catalyst and ModelingToolkit to enable the numerical solution of the chemical master equation, while DelaySSAToolkit.jl [12] can accept Catalyst models as input to its SSAs that handle stochastic chemical kinetics models with delays. Another example of how Catalyst's flexibility enables its integration into the Julia ecosystem is that CRNs with polynomial ODEs (a condition that holds for pure mass action systems) can be exported as polynomials. This enables polynomial methods, such as homotopy continuation, to be employed on Catalyst models. Here, homotopy continuation (implemented by the HomotopyContinuation.jl Julia package) can be used to reliably compute all roots of a polynomial system [6]. This is an effective approach for finding multiple steady states of a system. While the presence of Hill functions generates rational polynomial systems, from these the numerator can be extracted, and homotopy continuation can still be used.

Catalyst models are compatible with a wide range of ancillary tools and methods

The Julia SciML open source organization supports a wide range of techniques for working with models and data, based around the IR that Catalyst produces. While the reactions that constitute a CRN are often known in developing a model, system parameters rarely are (these typically correspond to the reaction rates). A first step in analysing a model is identifiability analysis, where we determine when the parameters can be uniquely identified from the data [30]. This is enabled through the StructuralIdentifiability.jl package. In the next step, parameters can be fitted to data. This can be done using DiffEqParamEstim.jl, which provides simple functions that are easy to use. Alternatively, more powerful packages, like the Turing.jl Julia library for Bayesian analysis, offer increased flexibility for experienced users [13]. Fitting model structure to data is also possible using the DataDrivenDiffEq.jl SciML package, for which specific CRN support is currently a work in progress. Alternatively, unknown CRN structures can be simplified using neural networks, which are then trained on data.

Finally, there exists a large number of tools within SciML and the wider Julia ecosystem which can be employed on Catalyst models (Figure 5). System steady states can be computed using SteadyStateDiffEq.jl or the HomotopyContinuation.jl Julia package [6]. The BifurcationKit.jl Julia package can be used to compute bifurcation diagrams [45]. Options for displaying CRNs, either as network graphs (via Graphviz) or Latex formatted equations (via Latexify.jl), also exist.

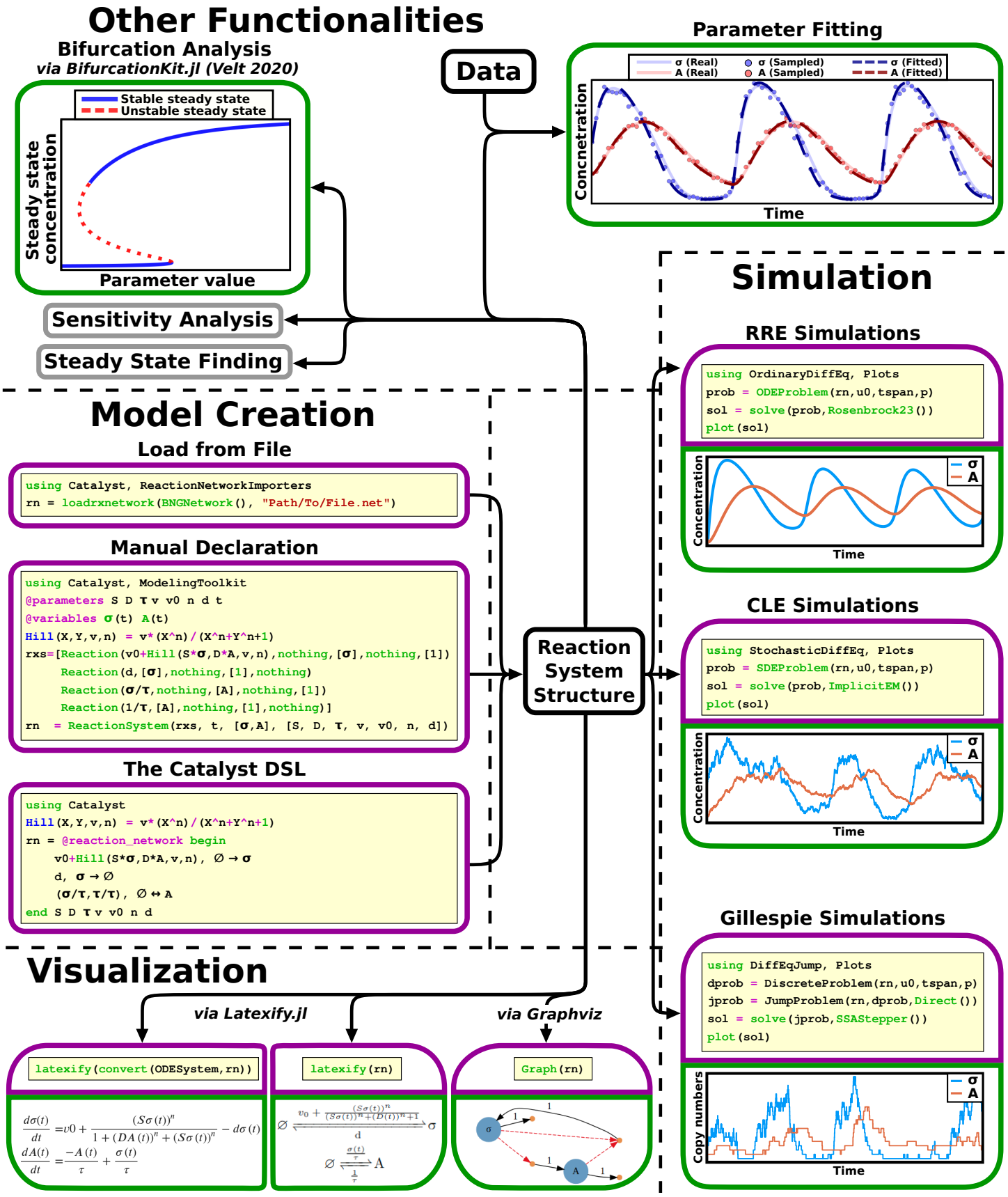

Figure 5: **A wide range of features are available for Catalyst model analysis.** A CRN model can be created either through the DSL, by manually declaring the reaction events, or by loading it from a file. The model is stored in the `ReactionSystem` IR, which can be used as input to a wide range of methods. Purple boxes indicate code written by the user, and green boxes the corresponding output. For some methods, either one, or both, boxes are omitted.

Discussion

In this article, we have introduced the Catalyst library for modeling of CRNs. It represents models through the ModelingToolkit.jl IR, which is ubiquitously used within the SciML organisation, and can generate optimized inputs for numerical simulations (RRE ODE, CLE SDE, and stochastic chemical kinetics jump process models). Moreover, it can compose with a variety of other Julia packages, including data-driven modeling tooling (parameter fitting and model inference), and other functionality (identifiable analysis, sensitivity analysis, steady state analys, etc). The IR is based on the Symbolics.jl CAS, enabling algebraic manipulation and simplification of Catalyst models. This can both be harnessed by the user (e.g. to create models programmatically) and by software (e.g. for automated Jacobian computations). Finally, this also enables easy connection to other Julia packages for symbolic analysis, such as enabling polynomial methods (e.g homotopy continuations) to act on CRN ODEs that have a polynomial form.

In addition to the wide range of powerful tools enabled by the combination of the ModelingToolkit IR and the Symbolics CAS, Catalyst also provides a DSL that simplifies the declaration of especially smaller models. Of a finalized pipeline that evaluates a model with respect to a specific scientific problem, the model declaration is typically only a minor part. However, reaching a final model often requires the production and analysis of several alternative network topologies. If the barrier to create or modify a model can be reduced, more topologies can be explored in a shorter time. Thus, an intuitive interface can greatly simplify the model exploration portion of a research project. By providing a DSL that reads CRN models in their most natural form, Catalyst helps to facilitate model construction. In addition, this form of declaration makes code easier to debug, as well as making it easier to understand for non-experts.

While several previous tools for CRN modeling have been primarily designed around their own interface, we have instead designed Catalyst to be called from within standard Julia programs and scripts. This is advantageous, since it allows the flexibility of analysing a model with custom code, without having to save and load simulation results to and from files. Furthermore, by integrating our tool into a larger context (SciML), support for a large number of higher-order features is provided, without requiring any separate implementation within Catalyst. This strategy, with modeling software targeting an IR (here provided by ModelingToolkit) enables modelers across widely different domains to collaborate in the development and maintenance of tools. We believe this is the ideal setting for a package like Catalyst.

Catalyst is available for free under the permissive MIT License. The source code can be found at https://github.com/SciML/Catalyst.jl. It is also a registered package within the Julia ecosystem and can be installed from within a Julia environment using the command `Pkg.add("Catalyst")`. Full documentation, including tutorials and an API, can be found at https://catalyst.sciml.ai/stable/. Issues and help requests can be raised either at the Catalyst GitHub page, on the Julia discourse forum (https://discourse.julialang.org/), or at the SciML organization's Julia language Slack channels (`#diffeq-bridged` and `#sciml-bridged`). The package is open to pull requests from anyone who wishes to contribute to its development. Users are encouraged to engage in the project. Current and future SciML organization efforts include the development of spatial partial differential equation, spatial jump process, τ-leaping, and hybrid/multiscale solvers, along with plans to extend the Catalyst interface to allow the specification of spatial and hybrid CRN models.

Methods

Benchmarks

Benchmarks were carried out using the five CRN models used in [20]. The .bngl files provided in [20] were used as input to BioNetGen, while COPASI, gillespy2, and Matlab used the corresponding (BioNetGen generated) .xml files. Catalyst used the corresponding (BioNetGen generated) .net files. BioNetGen, COPASI, and gillespy2 simulations were performed using their corresponding Python interfaces.

First, each model was simulated using each tool, and the corresponding solution trajectories were inspected, ensuring that the correct output was produced. Next, simulation run times were measured for a range of physical model final times. Runtimes were measured using `timeit` (in Python), BenchmarkTools.jl (in Julia, [8]), and `timeit` (in Matlab). The median runtime over several simulations was used for the plots in Figure 3. For ODE simulations, for all tools, the absolute tolerance was set to 10^{-12} and the relative tolerance 10^{-6}. Catalyst has access to additional ODE solvers via DifferentialEquations.jl, more specifically OrdinaryDiffEq.jl, and is able to attain even faster run times for some problems (such as QNDF and TRBDF2, [24, 37]). However, to ensure the benchmarks are comparable across the various tools, we limited ourselves to the CVODE and lsoda solvers, which the other tools rely on [33, 22]. In this way the ODE solver comparisons help measure the computational performance of the ODE derivative functions generated by Catalyst and ModelingToolkit in comparison to those of other packages. For the larger models, some packages were omitted from the reported benchmarks, as these were unable to successfully simulate the model within a reasonable time frame.

Model:	**Multistate**	**Multisite2**	**EGfr_net**	**BCR**	**Fceri_gamma2**
Explicit Jacobian	No	No	Yes	Yes	Yes
Sparse	No	No	Yes	Yes	Yes
Linear solver	dense LU	dense LU	KLU	KLU	KLU

Table 1: When using the CVODE solver with the larger EGfr_net, BCR, and Fceri_gamma2 networks, a sparse Jacobian evaluation function was automatically constructed for Catalyst models when generating the compiled ODE derivative functions with ModelingToolkit. This was found to reduce CVODE's overall simulation time. The KLU linear solver [10] was then used within CVODE. For the Multistate and Multisite2 networks, no explicit Jacobian evaluation function was provided to CVODE. In this case CVODE internally approximates the Jacobian using finite differences, and uses dense LU factorizations in solving linear systems.

In generating compiled functions from Catalyst models for use in ODE solvers, users can also choose to automatically generate functions for evaluating the Jacobian of the ODE derivative function, and can choose to automatically construct this function to work with a dense or sparse representation of the Jacobian matrix. With CVODE, we found that including a sparse Jacobian generally reduced simulation times for the larger reaction network models. Via DifferentialEquations.jl, one also has flexibility in their choice of linear solver to use within implicit methods. In our benchmarking, Jacobians were only used for the CVODE solver with Catalyst-generated models; the Jacobian and linear solver choices selected for this solver for each network are specified in Table 1.

Stochastic chemical kinetics simulations from Catalyst models used SSAs defined in JumpProcesses.jl [25], a component of DifferentialEquations.jl. In Figure 3, Direct refers to Gillespie's direct method [14], SortingDirect to the sorting direct method of [29], RSSA and RSSACR to the rejection SSA methods of [42, 43, 44]. Dependency graphs needed for the different methods are automatically generated via Catalyst and ModelingToolkit as input to the JumpProcesses.jl solvers.

The benchmarks were carried out on Julia version 1.7.3, using Catalyst version 11.0.0, JumpProcesses (formerly DiffEqJump) version 8.3.0, and OrdinaryDiffEq version 6.11.2. Note that JumpProcesses and OrdinaryDiffEq are both components in the meta DifferentialEquations.jl package. We used the version 0.7.0 python interface for BioNetGen, the basico version 0.16 python interface for Copasi, gillespy2 version 1.6.8, and Matlab version 9.11 with SimBiology version 6.2.

Code availability

Scripts for generating all figures presented here, as well as for carrying out the benchmarks, can be found at https://github.com/TorkelE/Catalyst-Fast-Biochemical-Modeling-with-Julia.

Acknowledgements

TL's contribution to this project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No.721456.

SAI's and CR's work on this project has been made possible in part by grants to the SciML organization; Wellcome Trust grant [223770/Z/21/Z], and grant number 2021-237457 from the Chan Zuckerberg Initiative DAF, an advised fund of Silicon Valley Community Foundation. SAI was also partially supported by NSF-DMS 1902854. VI was partially supported by a 2021 Google Summer of Code Fellowship and the Boston University UROP program.

CR's contribution to this material is based upon work supported by the National Science Foundation under grant no. OAC-1835443, grant no. SII-2029670, grant no. ECCS-2029670, grant no. OAC-2103804, and grant no. PHY-2021825. We also gratefully acknowledge the U.S. Agency for International Development through Penn State for grant no. S002283-USAID. The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0001211 and DE-AR0001222. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Agreement No HR00112290091. We also gratefully acknowledge the U.S. Agency for International Development through Penn State for grant no. S002283-USAID. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. This material was supported by The Research Council of Norway and Equinor ASA through Research Council project "308817 - Digital wells for optimal production and drainage". Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or

implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] John H. Abel, Brian Drawert, Andreas Hellander, and Linda R. Petzold. GillesPy: A Python Package for Stochastic Model Building and Simulation. *IEEE Life Sciences Letters*, 2(3):35–38, 2017.

[2] Dipak Barua, William S. Hlavacek, and Tomasz Lipniacki. A computational model for early events in b cell antigen receptor signaling: Analysis of the roles of lyn and fyn. *The Journal of Immunology*, 189(2):646–658, 2012.

[3] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[4] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Julia: A Fast Dynamic Language for Technical Computing. pages 1–27, 2012.

[5] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. *Biosystems*, 83(2):136–151, 2006. 5th International Conference on Systems Biology.

[6] Paul Breiding and Sascha Timme. Homotopycontinuation.jl: A package for homotopy continuation in julia. In James H. Davenport, Manuel Kauers, George Labahn, and Josef Urban, editors, *Mathematical Software – ICMS 2018*, pages 458–465, Cham, 2018. Springer International Publishing.

[7] Tom Breloff. Plots.jl, November 2021.

[8] Jiahao Chen and Jarrett Revels. Robust benchmarking in noisy environments. *arXiv e-prints*, Aug 2016.

[9] Joshua Colvin, Michael I. Monine, James R. Faeder, William S. Hlavacek, Daniel D. Von Hoff, and Richard G. Posner. Simulation of large-scale rule-based models. *Bioinformatics*, 25(7):910–917, 02 2009.

[10] Timothy A. Davis and Ekanathan Palamadai Natarajan. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. *ACM Trans. Math. Softw.*, 37(3), sep 2010.

[11] James R. Faeder, William S. Hlavacek, Ilona Reischl, Michael L. Blinov, Henry Metzger, Antonio Redondo, Carla Wofsy, and Byron Goldstein. Investigation of early events in fcεri-mediated signaling using a detailed mathematical model. *The Journal of Immunology*, 170(7):3769–3781, 2003.

[12] Xiaoming Fu, Xinyi Zhou, Dongyang Gu, Zhixing Cao, and Ramon Grima. DelaySSAToolkit.jl: Stochastic simulation of reaction systems with time delays in Julia. *Bioinformatics*, 07 2022.

[13] Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: a language for flexible probabilistic inference. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, pages 1682–1690, 2018.

[14] Daniel T. Gillespie. A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *Journal of Computational Physics*, 22:403–434, 1976.

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

[16] Daniel T. Gillespie. The chemical Langevin equation. *Journal of Chemical Physics*, 113(1):297–306, 2000.

[17] Ido Golding, Johan Paulsson, Scott M. Zawilski, and Edward C. Cox. Real-time kinetics of gene activity in individual bacteria. *Cell*, 123(6):1025–1036, 2005.

[18] A. Gonzalez Gonzalez, A. Naldi, L. Sánchez, D. Thieffry, and C. Chaouiya. GINsim: A software suite for the qualitative modelling, simulation and analysis of regulatory networks. *BioSystems*, 84(2):91–100, 2006.

[19] Shashi Gowda, Yingbo Ma, Alessandro Cheli, Maja Gwóźzdź, Viral B. Shah, Alan Edelman, and Christopher Rackauckas. High-performance symbolic-numerics via multiple dispatch. *ACM Commun. Comput. Algebra*, 55(3):92–96, jan 2022.

[20] Abhishekh Gupta and Pedro Mendes. An Overview of Network-Based and -Free Approaches for Stochastic Simulation of Biochemical Systems. *Computation (Basel)*, 6(1), 2018.

[21] Leonard A. Harris, Justin S. Hogg, José Juan Tapia, John A.P. Sekar, Sanjana Gupta, Ilya Korsunsky, Arshi Arora, Dipak Barua, Robert P. Sheehan, and James R. Faeder. BioNetGen 2.2: Advances in rule-based modeling. *Bioinformatics*, 32(21):3366–3368, 2016.

[22] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[23] Stefan Hoops, Ralph Gauges, Christine Lee, Jürgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. COPASI - A COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[24] ME Hosea and LF Shampine. Analysis and implementation of tr-bdf2. *Applied Numerical Mathematics*, 20(1-2):21–37, 1996.

[25] S. A. Isaacson, V. Ilin, and C. V. Rackauckas. JumpProcesses.jl. https://github.com/SciML/JumpProcesses.jl/, 2022.

[26] Steffen Klamt, Julio Saez-Rodriguez, and Ernst D Gilles. Structural and functional analysis of cellular networks with CellNetAnalyzer. *New York*, 13:1–13, 2008.

[27] Carlos F. Lopez, Jeremy L. Muhlich, John A. Bachman, and Peter K. Sorger. Programming biological models in Python using PySB. *Molecular Systems Biology*, 9(1):1–19, 2013.

[28] Yingbo Ma, Shashi Gowda, Ranjan Anantharaman, Chris Laughman, Viral Shah, and Chris Rackauckas. Modelingtoolkit: A composable graph transformation system for equation-based modeling, 2021. arXiv:2103.05244.

[29] J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Computational Biology and Chemistry*, 30(1), 2006.

[30] Hongyu Miao, Xiaohua Xia, Alan S. Perelson, and Hulin Wu. On identifiability of nonlinear ode models and applications in viral dynamics. *SIAM Review*, 53(1):3–39, 2011.

[31] N. Obatake, A. Shiu, X. Tang, and A. Torres. Oscillations and bistability in a model of ERK regulation. *J Math Biol*, 79(4):1515–1549, 09 2019.

[32] Kaan Öcal and Augustinas Sukys. FiniteStateProjection.jl. https://github.com/kaandocal/FiniteStateProjection.jl, 2022.

[33] Linda Petzold. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM Journal on Scientific and Statistical Computing*, 4(1):136–148, 1983.

[34] Christopher Rackauckas and Qing Nie. Adaptive methods for stochastic differential equations via natural embeddings and rejection sampling with memory. *Discrete Continuous Dyn Syst Ser B.*, 22(7):2731–2761, 2017.

[35] Christopher Rackauckas and Qing Nie. DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of Open Research Software*, 5(15):15, 2017.

[36] James Schaff, Charles C. Fink, Boris Slepchenko, John H. Carson, and Leslie M. Loew. A general computational framework for modeling cellular structure and function. *Biophysical Journal*, 73(3):1135–1146, 1997.

[37] Lawrence F Shampine and Mark W Reichelt. The matlab ode suite. *SIAM journal on scientific computing*, 18(1):1–22, 1997.

[38] Jorn Starruß, Walter De Back, Lutz Brusch, and Andreas Deutsch. Morpheus: A user-friendly modeling environment for multiscale and multicellular systems biology. *Bioinformatics*, 30(9):1331–1332, 2014.

[39] Melanie I. Stefan, Thomas M. Bartol, Terrence J. Sejnowski, and Mary B. Kennedy. Multi-state modeling of biomolecules. *PLOS Computational Biology*, 10(9):1–9, 09 2014.

[40] Augustinas Sukys and Ramon Grima. MomentClosure.jl: automated moment closure approximations in Julia. *Bioinformatics*, 38(1):289–290, 06 2021.

[41] Camille Terfve, Thomas Cokelaer, David Henriques, Aidan MacNamara, Emanuel Goncalves, Melody K. Morris, Martijn van Iersel, Douglas A. Lauffenburger, and Julio Saez-Rodriguez. CellNOptR: A flexible toolkit to train protein signaling networks to data using multiple logic formalisms. *BMC Systems Biology*, 6, 2012.

[42] Vo Hong Thanh, Corrado Priami, and Roberto Zunino. Efficient rejection-based simulation of biochemical reactions with stochastic noise and delays. *The Journal of Chemical Physics*, 141(13):134116–134113, 2014.

[43] Vo Hong Thanh, Roberto Zunino, and Corrado Priami. On the rejection-based algorithm for simulation and analysis of large-scale reaction networks. *The Journal of Chemical Physics*, 142(24):244106–244114, 2015.

[44] Vo Hong Thanh, Roberto Zunino, and Corrado Priami. Efficient constant-time complexity algorithm for stochastic simulation of large reaction networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 14(3):657–667, 2017.

[45] Romain Veltz. BifurcationKit.jl. https://hal.archives-ouvertes.fr/hal-02902346, Jul 2020. Package version: 0.1.8.

[46] José M.G. Vilar, Hao Yuan Kueh, Naama Barkai, and Stanislas Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Sciences of the United States of America*, 99(9):5988–5992, 2002.