The systems biology simulation core algorithm

Roland Keller[1], Alexander Dörr[1], Akito Tabira[2], Akira Funahashi[2], Michael J. Ziller[3], Richard Adams[4], Nicolas Rodriguez[5], Nicolas Le Novère[6], Hannes Planatscher[7], Andreas Zell[1], and Andreas Dräger[1*]

[1]Center for Bioinformatics Tuebingen (ZBIT), University of Tuebingen, Tübingen, Germany [2]Keio University, Graduate School of Science and Technology, Yokohama, Japan [3]Department of Stem Cell and Regenerative Biology, Harvard University, Cambridge, MA, USA [4]SynthSys Edinburgh, CH Waddington Building, University of Edinburgh, Edinburgh EH9 3JD, UK [5]European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge, UK [6]Babraham Institute, Babraham, Cambridge, UK [7]Natural and Medical Sciences Institute at the University of Tuebingen, Reutlingen, Germany

Email: Andreas Dräger - andreas.draeger@uni-tuebingen.de;

*Corresponding author

Abstract

Background: With the increasing availability of high dimensional time course data for metabolites, genes, or fluxes the description of dynamical systems by differential equation models becomes more and more popular in systems biology. Models are often encoded in formats such as SBML (Systems Biology Markup Language), whose mathematical structure is very complex and difficult to evaluate due to many special cases.

Results: This article describes an efficient algorithm to interpret and solve differential equation systems in SBML models. We begin our consideration with a formal representation of the mathematical form of the models and explain all parts of the algorithm in detail, including several pre-processing steps. We provide a flexible reference implementation of the algorithm as part of the Systems Biology Simulation Core Library, a community-driven project providing a large collection of numerical solvers and a sophisticated interface hierarchy for the definition of custom differential equation systems. To demonstrate the capabilities of the new algorithm, it has been tested with the entire SBML Test Suite and all models of BioModels Database.

Conclusions: The formal description of the mathematics behind the SBML format facilitates the implementation of the algorithm within specifically tailored programs. The reference implementation can be used as a simulation backend for Java™-based programs. Source code, binaries, and documentation can be freely obtained under the terms of the LGPL version 3 from http://sourceforge.net/projects/simulation-core/. Feature

requests, bug reports, contribution, or any further discussion can be directed to the mailing list simulation-core-development@lists.sourceforge.net.

Background

As part of the movement towards quantitative biology, modeling, simulation, and computer analysis of biological networks have become integral parts of modern biological research [1]. Ambitious national and international research projects such as the Virtual Liver Network [2] strive to derive even organ-wide models of biological systems that include all kinds of processes taking place at several levels of detail. Large-scale efforts like this require a strong collaboration between various research groups, including experimenters and modelers. The exchange, storage, interoperability and the possibility to combine models have been recognized as key aspects of this endeavor [3].

XML-based standard description formats [4] such as the Systems Biology Markup Language (SBML, [5]) or CellML [6] enable encoding of quantitative biological network models. To facilitate sharing and reuse of the models, online data bases such as the BioModels database [7] or the CellML model repository [8] provide large collections of published models in various formats. Software libraries for reading and manipulating the content of these formats are also available [9–11] and a large variety of programs supports these model description languages.

The models encoded in these formats can be interpreted in terms of differential equation systems, with additional structures such as discrete events and algebraic equations. The diversity of modeling aproaches and experimental data often requires customized software solutions for very specific tasks. For efficient model analysis, simulation, and calibration (e.g., the estimation of parameter values) a multiple-purpose and efficient numerical solver library that has been designed with the requirements of biological network models in mind is prerequisite. Although the language specifications of SBML [12–18] and CellML [19] describe the semantics of models in these formats and their interpretation, the algorithmic implementation is still not straightforward.

Both communities offer standardized and manually derived benchmark test in order to evaluate the quality of simulation results, because it has been recognized that in many cases different solver implementations lead to divergent results [20]. In this work we address the question of how to precisely interpret models encoded in the SBML format. Furthermore, we show how to adapt existing integration routines in order to simulate SBML models. To this end, we derive a new algorithm for the precise interpretation and simulation of all

currently existing levels and versions of SBML. As a reference implementation, we introduce an exhaustive implementation in the Java™ programming language. The algorithm described in this paper is, however, not limited to any particular programming language. It is also important to note that the interpretation of these models must be strictly separated from the numerical method that solves the implicated differential equation system. In this way, a similar approach would also be possible for further systems biological community formats, such as CellML.

The Systems Biology Simulation Core Library (SCL) presented here is a platform-independent, well-tested generic library that is completely decoupled from any graphical user interface and can therefore easily be integrated into third-party programs. It comprises several Ordinary Differential Equation (ODE) solvers and an interpreter for SBML models. It is the first simulation library based on JSBML [11]. Furthermore, the Systems Biology Simulation Core Library contains classes to both export simulation configurations to SED-ML (Simulation Experiment Description Markup Language, [21]), and facilitate the re-use and reproduction of these experiments by executing SED-ML files.

Results and discussion

In order to derive an algorithm for the interpretation of SBML models, it is first necessary to take a closer look at the mathematical equations implied by this data format. Based on this general description, we will then discuss all necessary steps to deduce an algorithm that takes all special cases for the various levels and versions of SBML into account.

A formal representation of models in systems biology

The mathematical structure of a reaction network comprises a stoichiometric matrix $\mathbf{N}$, whose rows correspond to the reacting species $\vec{S}$ within the system, whereas its columns represent the reactions, i.e., bio-transformations, in which these species participate. The velocities $\vec{\nu}$ of the reactions $\vec{R}$ determine the rate of change of the species' amounts:

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{S} = \mathbf{N}\vec{\nu}(\vec{S}, t, \mathbf{N}, \mathbf{W}, \vec{p})\,. \tag{1}$$

The parameter vector $\vec{p}$ contains rate constants and other, often constant, quantities that influence the reaction velocities. According to [22, 23] the modulation matrix $\mathbf{W}$ is defined as a matrix of size $|\vec{R}| \times |\vec{S}|$ containing the type of the regulatory influences of the species on the reactions, e.g., competetive inhibition or physical stimulation. Integrating Equation 1 yields the predicted amounts of the species at each time

point of interest within the interval $[t_0, t_T]$:

$$\vec{S} = \int_{t_0}^{t_T} \mathbf{N}\vec{\nu}(\vec{S}, t, \mathbf{N}, \mathbf{W}, \vec{p})\mathrm{d}t\,. \tag{2}$$

Depending on the units of the species, the same notation can also express the change of the species' concentrations. In this simple case, solving the homogeneous ordinary differential equation system 1 can be done in a straightforward way using many (numerical) differential equation solvers. The non-linear form of the kinetic equations in the vector function $\vec{\nu}$ constitutes the major difficulty for this endeavor and is often the reason why an analytical solution of these systems is not possible or very hard to achieve. Generally, differential equation systems describing biological networks are, however, inhomogenious systems with a higher complexity. Solving systems encoded in SBML can be seen as computing the solution of the following equation:

$$\vec{Q} = \int_{t_0}^{t_T} \mathbf{N}\vec{\nu}(\vec{Q}, t, \mathbf{N}, \mathbf{W}, \vec{p})\mathrm{d}t + \int_{t_0}^{t_T} \vec{g}(\vec{Q}, t)\mathrm{d}t + \vec{f}_E(\vec{Q}, t) + \vec{r}(\vec{Q}, t)\,. \tag{3}$$

The vector $\vec{Q}$ of quantities contains the sizes of the compartments $\vec{C}$, amounts (or concentrations) of reacting species $\vec{S}$, and the values of all global model parameters $\vec{P}$. It should be noted that these models may contain local parameters $\vec{p}$ that influence the reaction velocities, but which are not part of the global parameter vector $\vec{P}$, and hence also not part of $\vec{Q}$. All vector function terms may involve a delay function, i.e., an expression of the form delay(x, τ) with $\tau > 0$. In this way, it is possible to address values of x computed in the earlier integration step at time $t - \tau$, turning Equation 3 into a delay differential equation.

In the general case of Equation 3, not all species' amounts can be computed by integrating the transformation $\mathbf{N}\vec{\nu}$: The change of some model quantities may be given in form of rate rules (function $\vec{g}(\vec{Q}, t)$) that must be integrated separately. Species, whose amounts are determined by rate rules, must not participate in any reaction and do hence have only zero-valued corresponding entries in the stoichiometric matrix $\mathbf{N}$. Thereby, the rate rule function $\vec{g}(\vec{Q}, t)$ directly gives the rate of change of these quantities, and returns 0 for all others.

In addition, SBML introduces the concept of events $\vec{f}_E(\vec{Q}, t)$ and assignment rules $\vec{r}(\vec{Q}, t)$. An event can directly manipulate the value of several quantities, for instance, reduce the size of a compartment to a certain portion of its current size, as soon as a trigger condition becomes satisfied. An assignment rule also influences the absolute value of some quantity.

A further concept of SBML models are algebraic rules, which are equations that must evaluate to zero at all times during the simulation of the model. These rules can be solved to determine the values of quantities, whose values are not determined by other constructs. In this way, conservation relations or other complex

interrelations can be expressed in a very convenient way. With the help of a bipartite matching and a subsequent conversion it is possible to turn algebraic rules into assignment rules and hence include these into the term $\vec{r}(\vec{Q}, t)$. Such a transformation, however, requires symbolic computation and is hence a complicated endeavor.

In case that the system under study operates at multiple time scales, i.e., it contains a fast and a slow sub-system, a separation of the system is necessary leading to differential algebraic equations. Some species can be declared to operate at the system's boundaries, assuming a constant pool of their amounts or concentrations. Care must also be taken with respect to the units of the species, because under certain condition division or multiplication with the sizes of their surrounding compartments is necessary in order to ensure the consistent interpretation of the models. For all these reasons, solving Equation 3 is much more complicated than computing the solution of the simple Equation 1 alone.

From the perspective of software engineering, a strict separation of the interpretation of the model and the numerical treatment of the differential equation system is necessary to ensure that regular numerical methods can be used to solve Equation 3. In order to efficiently compute this solution, multiple pre-processing steps are required, such as the conversion of algebraic rules into assignment rules, or avoiding repeated re-computation of intermediate results. The next sections will give a detailed explanation of the necessary steps to solve these systems and how to efficiently perform their numerical integration with standard numerical solvers.

Initialization

At the beginning of the simulation the values of species, parameters and compartments are set to the initial values as given in the model. All kinetic laws of the reactions, assignment rules, transformed algebraic rules (see below), initial assignments, event assignments and rate rules are integrated into one directed acyclic syntax graph. This graph is hence the result of merging of the abstract syntax trees representing all those individual elements. Equivalent elements are only contained once. This significantly decreases the computation time needed for the evaluation of these syntax graphs during the simulation, in comparison to maintaining multiple syntax trees. Figure 1 gives an example for such a syntax graph.

After the creation of the abstract syntax graph, the initial assignments, and the assignment rules (including transformed algebraic rules) are processed. Deviating initial values defined by initial assignments and are now assigned.

Solving algebraic rules

In order to deal with algebraic rules in an SBML model, these have to be converted to assignment rules. In every equation of an algebraic rule, there should be at least one variable, whose value is not yet defined through other equations in model. This variable has to be determined for the purpose of interpreting the regarding algebraic rule. At first, a bipartite graph according to the SBML specifications is generated. This graph is used to compute a matching, using the algorithm by Hopcroft and Karp [24]. The initial greedy matching is extended with the use of augmenting paths. This process is repeated until no more augmenting paths can be found. Per definition, this results in a maximal matching. As stated in the SBML specifications, if any equation vertex remains unconnected after augmenting the matching as far as possible, the model is considered overdetermined and thus is not a valid SBML model. If this is not the case, every assignment rule is now converted into an algebraic rule. To this end, the mathematical expression is transformed into an equation with the target variable on its left-hand side. The left-hand side is represented by the respective variable vertex, to which the considered algebraic rule has been matched. Figure 2 displays the described algorithm in form of a flow chart.

Event handling

An event in SBML is a function that is executed depending on whether a trigger condition switches from *false* to *true*. In addition, a delay may postpone the actual execution of the event's assignments to a later point in time. With the release of SBML Level 3 Version 1, the processing of events has been raised to a higher level of complexity. Before Level 3 Version 1 it was sufficient to determine, when an event triggers and when its assignmens are to be exceuted. In Level 3 Version 1 just a few new language elements have been added but with an huge impact on how to handle events: The order in which events have been processed used to be at programmers discretion, but now it is given by the event's priority element. Coordinating the sequence, in which events are to be executed, has now become the crucial part of event handling. Furthermore, there exists the option to cancel an event during the time since its trigger has been activated and the actual time when the scheduler picks the event for execution.

For every time step, the events to be executed are a union of two subsets of the set of all events. On the one hand, there are the events, whose trigger has been activated at the given time and which are to be evaluated without delay. On the other hand, there are events that triggered at some time point before, and whose delay reaches till the current point in time. For every element of the resulting set of events their priority rule must be evaluated. One event is randomly chosen for execution from all events with the

highest priority. All other events could be handled in the same manner. However, the assignment of the first event can change the priority or even the trigger condition of the events that have not yet been executed. Therefore, the trigger of non persistent events and the priority of the remaining events have to be evaluated again. In this case, another event that has now the highest priority is chosen. This process repeats itself until no further events are left to be executed. Figure 3 shows the slightly simplified algorithm for event processing at a specific point in time.

Integrated calculation for a certain time step including event processing

The precise calculation of the time, when events are triggered is crucial to ensure exact results of the numerical integration process. It could, for instance, happen that an event is triggered at time t_τ, which is between the two integration time points $t_{\tau-1}$ and $t_{\tau+1}$. When processing the events only at time points $t_{\tau-1}$ and $t_{\tau+1}$ it might happen that the trigger condition cannot be evaluated to *true* at neither of these time points. The Rosenbrock solver [25] can adapt its step size h if events occur (see Figure 4 for details). For a certain time interval $[t_{\tau-1}, t_{\tau+1}]$ and the current vector $\vec{Q}$ Rosenbrock's method determines the new value of vector $\vec{Q}$ at a point in time $t_{\tau-1} + h$. If the error tolerance cannot be ensured, h is reduced and the procedure is repeated.

After that the events and the assignment rules are processed at the new point in time $t_{\tau-1} + h$. If the previous step causes a change in $\vec{Q}$, the adaptive step size is decreased by setting h to $h/10$ and the calculation is repeated until the minimum step size is reached or the processing of events and rules does not change $\vec{Q}$ anymore. Hence, the time, at which an event takes place, is precisely determined.

Calculation of the derivatives for a certain point in time

For given values $\vec{Q}$ at a point t in time the current vector of derivatives $\dot{\vec{Q}}$ is calculated as follows: First, the rate rules are processed $\dot{\vec{Q}} = \vec{g}(\vec{Q}, t)$. Note that function $\vec{g}$ returns 0 in all dimensions, in which no rate rule is defined. Second, the velocity ν_i of each reaction R_i is computed with the help of the unified syntax graph (Figure 1 shows an example of such a graph). The velocity functions depend on $\vec{Q}$ at time t. During the second step the derivatives of all species that participate in the current reaction R_i need to be updated (see the flowchart in Figure 5).

A reference implementation of the algorithm as part of the Systems Biology Simulation Core Library

The algorithm described above has been implemented in Java™ and included into the Systems Biology Simulation Core Library. Figure 6 displays the software architecture of this library, which has been designed as part of a community project aiming to provide an extensible numerical backend for customized programs for research in computational systems biology. The SBML-solving algorithm is based on the data structures provided by the JSBML project [26]. With the help of wrapper classes several numerical solvers originating from the Apache Commons Math library could be included into the project. In addition, the library provides an implementation of the explicit fourth order Runge-Kutta method, Rosenbrock's solver, and Euler's method.

Each solver has a method to directly access its corresponding KiSAO (Kinetic Simulation Algorithm Ontology) term [27]. Due to the strict separation between numerical differential equation solvers, and the definition of the actual differential equation system, it is possible to implement support for further community standards, such as CellML [6].

In order to support the emerging standard MIASE (Minimal Information About a Simulation Experiment) [28], the library also provides an interpreter of SED-ML files (Simulation Experiment Description Markup Language) [21]. These files allow users to store the details of a simulation, including the selection and all settings of the numerical method, hence facilitating the creation of reproducible results. A simulation experiment can also be directly started by passing a SED-ML file to this interpreter.

Many interfaces, abstract classes, and an exhaustive source code documentation in form of JavaDoc facilitate the customization of the library. For testing purposes, the library contains a sample program that benchmarks the SBML interpreter against the entire SBML Test Suite.

Application to published models

The Systems Biology Simulation Core Library has been tested on all 424 curated models from BioModels database [7] (release 23, October 2012). As a result, 99.06 % of these models could be correctly simulated. In the following, we use two models from this repository to illustrate the capabilities of this library: Biomodel 206 by Wolf *et al.* [29] and Biomodel 390 by Arnold and Nikoloski [30].

The model by Wolf *et al.* [29] mimics glycolytic oscillations that have been observed in yeast cells. The model describes how the dynamics propagate through the cellular network. Figure 7a displays the simulation results for the intracellular concentrations of 3-phosphogylcerate, ATP, glucose, glyceraldhyde 3-phosphate, and NAD^+: After an initial phase of approximately 15 s all metabolites begin a steady-going rhythmic

oscillation. Changes in the dynamics of the fluxes through selected reaction channels within this model can be seen in Figure 7b.

By comparing a large collection of previous models of the Calvin-Benson cycle, Arnold and Nikoloski created a quantitative consensus model that comprises eleven species, six reactions, and one assignment rule [30]. All kinetic equations within this model call specialized function definitions. Figure 8 shows the simulation results for the species ribulose 1,5 bisphosphate, ATP, and ADP within this model. As in the previous test case, the simulation results reproduce the values provided by BioModels database.

Comparison to existing solver implementations for SBML

In order to benchmark our software, we chose similar tools exhibiting the following features from the SBML software matrix[1]:

- The last updated version was released after the final release of the specification for SBML Level 3 Version 1 Core, i.e., October 6$^{\text{th}}$ 2010.
- Open-source software
- No dependency on commercial products that are not freely available (e.g., MATLAB™ or Mathematica™)
- Support for SBML Level 3.

The selected programs are in alphabetical order: BioUML [31, 32], COPASI [33], iBioSim [34], JSim [35], LibSBMLSim [36], and VCell [37]. Table 1 summarizes the comparison of all six programs.

Conclusions

The SBML implementation has successfully passed the SBML Test Suite (version 2.0.2) using Rosenbrock's solver. The results are shown in Table 2.

All models together can be simulated within seconds, which means that the simulation of one SBML model takes only milliseconds on average, using regular desktop computers. The total simulation time for all models in SBML Level 3 Version 1 is significantly higher than for the models in other SBML levels and versions. This can be explained by the fact that there are some models in SBML Level 3 Version 1, in which a time-consuming processing of a large number of events is necessary. In particular, the simulation of

[1] http://sbml.org/SBML_Software_Guide/SBML_Software_Matrix (October 8$^{\text{th}}$ 2012)

model 966 of the SBML Test Suite, which is only provided in SBML Level 3 Version 1, takes 21 s because it contains 23 events to be processed. The evaluation of this model accounts for 60 % of the total simulation time for the models in SBML Level 3 Version 1. Furthermore, the Systems Biology Simulation Core Library solves 99.06 % of the models from the BioModels.net database (release 23, [7]) simulating from $t = 0$ to $t = 10$ using Rosenbrock solver and a step size of 0.1. These results suggest the reliability of the simulation algorithm described in this work.

Our tests indicate that only two programs pass the entire Test Suite for all SBML levels and versions: BioUML, which is a workbench for modelling, simulation, and parameter fitting, and the Systems Biology Simulation Core Library. The Systems Biology Simulation Core Library is therefore the only API simulation library exhibiting this capability.

Therefore, the Systems Biology Simulation Core Library is an efficient Java tool for the simulation of differential equation systems used in systems biology. It can be easily integrated into larger customized applications. For instance, CellDesigner version 4.2 [38] already uses it as one of its simulation libraries. The stand-alone application SBMLsimulator[2] provides a convenient graphical user interface for the simulation of SBML models and uses it as a computational backend. The abstract class structure of the library supports the integration of additional model formats, such as CellML, besides its SBML implementation. To this end, it is only necessary to implement a suitable interpreter class.

By including support for the emerging standard SED-ML, we hope to facilitate the exchange, archival and reproduction of simulation experiments performed using the Systems Biology Simulation Core Library.

Methods

Implementation

All the solver classes are derived from the abstract class `AbstractDESSolver` (Figure 6). Several solvers of the Apache Commons Math library (version 3.0) are integrated with the help of wrapper classes. Numerical methods and the actual differential equation systems are strictly separated. The class `MultiTable` stores the results of a simulation within its `Block` data structures. The abstract description of differential equation systems, with the help of several distinct interfaces, makes possible to decouple them from a particular type of biological network. It is therefore possible to pass an instance of an interpreter for a respective model description format to any available solver. This interpretation is the most time consuming step of the integration procedure. This is why efficient and clearly organized data structures are required to ensure

[2]SBMLsimulator is available at http://www.cogsys.cs.uni-tuebingen.de/software/SBMLsimulator.

a high performance of the overall library. The interpretation of SBML models is split between evaluation of events and rules, computation of stoichiometric information, and computation of the current values for all model components (such as species and compartments). For a given state of the ODE system, the class `SBMLinterpreter`, responsible for the evaluation of models encoded in SBML returns the current set of time-derivatives of the variables. It is connected to an efficient MathML interpreter of the expressions contained in kinetic laws, rules and events (`ASTNodeInterpreter`). The nodes of the syntax graph for those expressions depend on the current state of the ODE system. If the state has changed, the values of the nodes have to be recalculated (see Results).

An important aspect in the interpretation of SBML models is the determination of the exact time at which an event occurs, as this influences the precision of the system's variables. To this end, we adjusted the Rosenbrock solver [39], an integrator with an adaptive step size, to a very precise timing of the events. In addition to events, rules are also treated during integration. Basically, rules are events that occur at every given point in time and are therefore processed in the same manner. For every object of the type `AlgebraicRule`, a new `AssignmentRule` object is generated by means of the preceding bipartite matching. They represent only temporary rules, that are incorporated in the simulation process but do not influence the model in the SBML file.

In the `SBMLinterpreter` events are represented via an array containing an object of the class `EventInProgress` for every event in the model. Thereby the distinction between events with and without delays is made. The major difference between both is that an event with delay can trigger multiple times before it is executed. In order to deal with such an issue, the class `SBMLEventInProgressWithDelay` keeps track of this via the help of a list containing the points in time, at which the respective event has to executed. When events trigger more than once before execution, they have to be ordered according to their delay because the delay of the very same event may vary.

When the `SBMLinterpreter` is processing events with priority, the events with the highest priority are currently stored in list until one of them is selected for execution. One could argue, that all events can be kept in the same data structure, e.g., in a binary max heap, where after the extraction of the element with the largest value, the heap is restructured so that that next largest value is at the top. As stated in the Results section, the execution of one event can influence the priority off the remaining events. Considering the binary max heap, there is the possibility that many priorities change whereby the standard method to restore the max heap characteristic after extraction is not sufficient any more. Therefore, we disregarded the use of other data structures for now.

SED-ML support is enabled by inclusion of the `jlibsedml` library (http://www.jlibsedml.org) in the binary download. Clients of the the Systems Biology Simulation Core Library can choose to use the `jlibsedml` API directly, or access SED-ML support via facade classes in the `org.simulator.sedml` package that do not require direct dependencies on `jlibsedml` in their code.

Availability and Requirements

The current version of Systems Biology Simulation Core Library is available on the project homepage. The entire project, including source code and documentation, several versions of jar files containing only binaries, binaries together with source code, can be downloaded, optionally also as a version including all required third-party libraries.

Project name: Systems Biology Simulation Core Library (SCL)

Project homepage: http://sourceforge.net/projects/simulation-core/

Operating Systems: Platform independent, i.e., for all systems for which a JVM is available. The Systems Biology Simulation Core Library was successfully tested under Linux (Ubuntu version 10.4), Mac OS X (versions 10.6.8 and 10.8.2), and Windows 7.

Programming Language: Java™

Other Requirements Java Runtime Environment 1.6 or above

License: LGPL version 3

Author's contributions

RK and AlD contributed equally, implemented the majority of the source code, and declare shared first authorship. AnD initialized and coordinated the project, drafted the manuscript, and supervised the work together with AZ. MJZ and HP designed and implemented the abstraction scheme between solvers and ODE systems. NR and NLN designed, implemented, and coordinated the data structures for a smooth integration of JSBML. RA implemented support for SED-ML. AT and AF incorporated the Simulation Core Library into CellDesigner. All authors contributed to the implementation, read and approved the final manuscript.

Acknowledgements

The authors are grateful to B. Kotcon, S. Mesuro, D. Rozenfeld, A. Yodpinyanee, A. Perez, E. Doi, R. Mehlinger, S. Ehrlich, M. Hunt, G. Tucker, P. Scherpelz, A. Becker, E. Harley, and C. Moore, Harvey Mudd College, USA, for providing a Java implementation of Rosenbrock's method, and to Michael T. Cooling, University of Auckland, New Zealand, for fruitful discussion. The authors thank D. M. Wouamba, P. Stevens, M. Zwießele, M. Kronfeld, and A. Schröder for source code contribution and fruitful discussion.

This work was funded by the Federal Ministry of Education and Research (BMBF, Germany) as part of the Virtual Liver Network (grant number 0315756).

References

1. Macilwain C: **Systems biology: evolving into the mainstream**. *Cell* 2011, **144**(6):839–841, [http://dx.doi.org/10.1016/j.cell.2011.02.044].
2. Holzhütter HG, Drasdo D, Preusser T, Lippert J, Henney AM: **The virtual liver: a multidisciplinary, multilevel challenge for systems biology**. *Wiley Interdiscip Rev Syst Biol Med* 2012, **4**(3):221–235, [http://dx.doi.org/10.1002/wsbm.1158].
3. Liebermeister W, Krause F, Uhlendorf J, Lubitz T, Klipp E: **SemanticSBML: a tool for annotating, checking, and merging of biochemical models in SBML format**. In *3rd International Biocuration Conference*, Nature Publishing Group 2009[http://dx.doi.org/10.1038/npre.2009.3093.1].
4. Bray T, J P, Sperberg-McQueen C, E M, Yergeau F: **Extensible Markup Language (XML) 1.0**. Tech. rep., W3C recommendation 2000.
5. Hucka M, Finney A, Bornstein BJ, Keating SM, Shapiro BE, Matthews J, Kovitz BL, Schilstra MJ, Funahashi A, Doyle JC, Kitano H: **Evolving a lingua franca and associated software infrastructure for computational systems biology: the Systems Biology Markup Language (SBML) project**. *Systems Biology, IEE* 2004, **1**:41–53, [http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1334988].
6. Lloyd CM, Halstead MDB, Nielsen PF: **CellML: its future, present and past**. *Prog Biophys Mol Biol* 2004, **85**(2-3):433–450, [http://dx.doi.org/10.1016/j.pbiomolbio.2004.01.004].
7. Le Novère N, Bornstein BJ, Broicher A, Courtot M, Donizelli M, Dharuri H, Li L, Sauro H, Schilstra M, Shapiro B, Snoep JL, Hucka M: **BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems**. *Nucleic Acids Research* 2006, **34**:D689–D691, [http://nar.oxfordjournals.org/cgi/content/full/34/suppl_1/D689].
8. Lloyd CM, Lawson JR, Hunter PJ, Nielsen PF: **The CellML Model Repository**. *Bioinformatics* 2008, **24**(18):2122–2123, [http://dx.doi.org/10.1093/bioinformatics/btn390].
9. Bornstein BJ, Keating SM, Jouraku A, Hucka M: **LibSBML: an API library for SBML**. *Bioinformatics* 2008, **24**(6):880–881, [http://dx.doi.org/10.1093/bioinformatics/btn051].
10. Miller AK, Marsh J, Reeve A, Garny A, Britten R, Halstead M, Cooper J, Nickerson DP, Nielsen PF: **An overview of the CellML API and its implementation**. *BMC Bioinformatics* 2010, **11**:178, [http://dx.doi.org/10.1186/1471-2105-11-178].
11. Dräger A, Rodriguez N, Dumousseau M, Dörr A, Wrzodek C, Le Novère N, Zell A, Hucka M: **JSBML: a flexible Java library for working with SBML**. *Bioinformatics* 2011, **27**(15):2167–2168, [http://bioinformatics.oxfordjournals.org/content/27/15/2167].
12. Hucka M, Finney A, Sauro H, Bolouri H: **Systems Biology Markup Language (SBML) Level 1: Structures and Facilities for Basic Model Definitions**. Tech. rep., Systems Biology Workbench Development Group JST ERATO Kitano Symbiotic Systems Project Control and Dynamical Systems, MC 107-81, California Institute of Technology, Pasadena, CA, USA 2001.
13. Hucka M, Finney A, Sauro H, Bolouri H: **Systems Biology Markup Language (SBML) Level 1: Structures and Facilities for Basic Model Definitions**. Tech. Rep. 2, Systems Biology Workbench Development Group JST ERATO Kitano Symbiotic Systems Project Control and Dynamical Systems, MC 107-81, California Institute of Technology, Pasadena, CA, USA 2003.
14. Finney A, Hucka M: **Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions**. Tech. rep., Systems Biology Workbench Development Group JST ERATO Kitano Symbiotic Systems Project Control and Dynamical Systems, MC 107-81, California Institute of Technology 2003.
15. Finney A, Hucka M, Le Novére N: **Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions**. Tech. rep. 2006.
16. Hucka M, Finney AM, Hoops S, Keating SM, Le Novére N: **Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions**. Tech. rep. 2007.
17. Hucka M, Finney A, Hoops S, Keating SM, Le Novère N: **Systems biology markup language (SBML) Level 2: structures and facilities for model definitions**. Tech. rep., Nature Precedings 2008, [http://dx.doi.org/10.1038/npre.2008.2715.1].

18. Hucka M, Bergmann FT, Hoops S, Keating SM, Sahle S, Schaff JC, Smith L, Wilkinson DJ: **The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core**. Tech. rep., Nature Precedings 2010, [http://precedings.nature.com/documents/4959/version/1].

19. Cuellar A, Nielsen P, Halstead M, Bullivant D, Nickerson D, Hedley W, Nelson M, Lloyd C: **CellML 1.1 Specification**. Tech. rep., Bioengineering Institute, University of Auckland 2006, [http://www.cellml.org/specifications/cellml_1.1/].

20. Bergmann FT, Sauro HM: **Comparing simulation results of SBML capable simulators**. *Bioinformatics* 2008, **24**(17):1963–1965, [http://bioinformatics.oxfordjournals.org/cgi/content/abstract/24/17/1963].

21. Waltemath D, Adams R, Bergmann FT, Hucka M, Kolpakov F, Miller AK, Moraru II, Nickerson D, Sahle S, Snoep JL, Le Novère N: **Reproducible computational biology experiments with SED-ML–the Simulation Experiment Description Markup Language**. *BMC Syst Biol* 2011, **5**:198, [http://dx.doi.org/10.1186/1752-0509-5-198].

22. Liebermeister W, Klipp E: **Bringing metabolic networks to life: convenience rate law and thermodynamic constraints**. *Theor Biol Med Model* 2006, **3**(42):41, [http://dx.doi.org/10.1186/1742-4682-3-41].

23. Liebermeister W, Uhlendorf J, Klipp E: **Modular rate laws for enzymatic reactions: thermodynamics, sensitivities, and implementation**. *submitted to Bioinformatics* 2010.

24. Hopcroft JE, Karp RM: **An n^5/2 algorithm for maximum matchings in bipartite graphs**. *SIAM Journal on Computing* 1973, **2**(4):225–231, [http://dx.doi.org/10.1137/0202019].

25. Press WH, Teukolsky SA, Vetterling WT, Flannery BP: *Numerical Recipes in FORTRAN; The Art of Scientific Computing*. NY, USA: Cambridge University Press 1993.

26. Dräger A, Hassis N, Supper J, Schröder A, Zell A: **SBMLsqueezer: a CellDesigner plug-in to generate kinetic rate equations for biochemical networks**. *BMC Systems Biology* 2008, **2**:39, [http://www.biomedcentral.com/1752-0509/2/39].

27. Courtot M, Juty N, Knüpfer C, Waltemath D, Zhukova A, Dräger A, Dumontier M, Finney A, Golebiewski M, Hastings J, Hoops S, Keating S, Kell DB, Kerrien S, Lawson J, Lister A, Lu J, Machne R, Mendes P, Pocock M, Rodriguez N, Villéger A, Wilkinson DJ, Wimalaratne S, Laibe C, Hucka M, Le Novère N: **Controlled vocabularies and semantics in systems biology**. *Mol Syst Biol* 2011, **7**:543, [http://dx.doi.org/10.1038/msb.2011.77].

28. Waltemath D, Adams R, Beard DA, Bergmann FT, Bhalla US, Britten R, Chelliah V, Cooling MT, Cooper J, Crampin EJ, Garny A, Hoops S, Hucka M, Hunter P, Klipp E, Laibe C, Miller AK, Moraru I, Nickerson D, Nielsen P, Nikolski M, Sahle S, Sauro HM, Schmidt H, Snoep JL, Tolle D, Wolkenhauer O, Le Novère N: **Minimum Information About a Simulation Experiment (MIASE)**. *PLoS Comput Biol* 2011, **7**(4):e1001122, [http://dx.doi.org/10.1371/journal.pcbi.1001122].

29. Wolf J, Passarge J, Somsen OJG, Snoep JL, Heinrich R, Westerhoff HV: **Transduction of intracellular and intercellular dynamics in yeast glycolytic oscillations**. *Biophys J* 2000, **78**(3):1145–1153, [http://dx.doi.org/10.1016/S0006-3495(00)76672-0].

30. Arnold A, Nikoloski Z: **A quantitative comparison of Calvin-Benson cycle models**. *Trends Plant Sci* 2011, **16**(12):676–683, [http://dx.doi.org/10.1016/j.tplants.2011.09.004].

31. Kolpakov FA, Tolstykh NI, Valeev TF, Kiselev IN, Kutumova EO, Ryabova A, Yevshin IS, Kel AE: **BioUML–open source plug-in based platform for bioinformatics: invitation to collaboration**. In *Moscow Conference on Computational Molecular Biology*, Department of Bioengineering and Bioinformatics of MV Lomonosov Moscow State University 2011:172–173.

32. Kolpakov FA, Puzanov M, Koshukov A: **BioUML: visual modeling, automated code generation and simulation of biological systems**. In *Proceedings of The Fifth International Conference on Bioinformatics of Genome Regulation and Structure*, Institute of Systems Biology, Novosibirsk 2006.

33. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, Singhal M, Xu L, Mendes P, Kummer U: **COPASI–a COmplex PAthway SImulator**. *Bioinformatics* 2006, **22**(24):3067–3074, [http://dx.doi.org/10.1093/bioinformatics/btl485].

34. Myers CJ, Barker N, Jones K, Kuwahara H, Madsen C, Nguyen NPD: **iBioSim: a tool for the analysis and design of genetic circuits**. *Bioinformatics* 2009, **25**(21):2848–2849, [http://dx.doi.org/10.1093/bioinformatics/btp457].

35. Raymond GM, Butterworth E, Bassingthwaighte JB: **JSIM: Free software package for teaching physiological modeling and research**. *Experimental Biology* 2003, **280**:102–107.

36. **LibSBMLSim: The library for simulating SBML models** [http://fun.bio.keio.ac.jp/software/libsbmlsim/].

37. Moraru II, Schaff JC, Slepchenko BM, Blinov ML, Morgan F, Lakshminarayana A, Gao F, Li Y, Loew LM: **Virtual Cell modelling and simulation software environment**. *IET Syst Biol* 2008, **2**(5):352–362, [http://dx.doi.org/10.1049/iet-syb:20080102].

38. Funahashi A, Tanimura N, Morohashi M, Kitano H: **CellDesigner: a process diagram editor for gene-regulatory and biochemical networks**. *BioSilico* 2003, **1**(5):159–162, [http://www.sciencedirect.com/science/article/B75GS-4BS08JD-5/2/5531c80ca62a425f55d224b8a0d3f702].

39. Kotcon B, Mesuro S, Rozenfeld D, Yodpinyanee A: *Final Report for Community of Ordinary Differential Equations Educators*. Harvey Mudd College Joint Computer Science and Mathematics Clinic, 301 Platt Boulevard, Claremont, CA 91711 2011, [http://www.math.hmc.edu/clinic/projects/2010/].

40. Hairer E, Nørsett SP, Wanner G: *Solving ordinary differential equations. 1, Nonstiff problems*. Berlin, Germany: Springer 2000.

Figures

Figure 1 - Example for the creation of an abstract syntax graph of a small model

The model consists of the following reactions:

$$R_1: \quad F1,6BP \rightleftharpoons DHAP + GA3P$$
$$R_2: \quad DHAP \rightleftharpoons GA3P$$

The two reactions are part of the glycolysis, the contained molecules are fructose 1,6-bisphosphate (F1,6BP), dihydroxyacetone phosphate (DHAP) and glyceraldehyde 3-phosphate (GA3P). Using the program SBMLsqueezer [26] the following mass action kinetics have been created:

$$\nu_{R_1} = k_{+1} \cdot [F16BP] - k_{-1} \cdot [DHAP] \cdot [GA3P]$$
$$\nu_{R_2} = k_{+2} \cdot [DHAP] - k_{-2} \cdot [GA3P]$$

The nodes for [DHAP] and [GA3P] are only contained in the syntax graph once and connected to more than one multiplication node. This figure clearly indicates that the syntax graph is not a tree.

Figure 2 - Algorithm for transforming algebraic rules to assignment rules

The first step is to decide whether the model is overdetermined by creating a matching of the variables. If this is not the case, every algebraic rule is solved to the matched variable, which provides the basis for the creation of an equivalent assignment rule.

Figure 3 - Processing of events: simpflified algorithm (handling of delayed events omitted)

Let E be the set of all events in a model, and I_E be the set of events, whose trigger conditions have already been evaluated to *true* in the previous time step. We refer to elements within I_E as *inactive* events. We define the set A_E as the subset of E containing those events, whose trigger condition switches from *false* to *true* in the current point t in time. At the beginning of the event handling, A_E is empty. We call an event *persistent* if it can only be removed from A_E under the condition that all of its assignments have been evaluated. This means that a *non-persistent* event can be removed from A_E in case that its trigger condition becomes *false* during the evaluation of other events. Here, the function trig(e) returns 0 or 1 depending on whether or not the trigger condition of event $e \in E$ is satisfied. Similarly, the function persist(e) returns 0 if event e is not persistent, or 1 otherwise. In each iteration the trigger conditions of those active events $a_e \in A_E$ that are not persistent are checked. If the trigger condition of such an event has changed from *true* (1) to *false* (0), it is removed from A_E. The next step comprises the evaluation of the triggers of all events. If its trigger changes from *false* to *true*, an event is added to the set of active events A_E. An event with its trigger changed from *true* to *false* is removed from the list of inactive events. After the procession of all triggers the event e of highest priority in the set of active events is chosen for execution by the function choose(A_E) . Note that priorities are not always defined, or multiple events may have an identical priority. The function choose(A_E) is therefore more complex than can be shown in this figure. This event is then processed, i.e., all of its assignments are evaluated, and afterwards the triggers of all events in E have to be evaluated again, because of possible mutual influences between the events. The algorithm proceeds until the list of acitve events is empty.

Figure 4 - Integrated calculation of new values for a time step including event processing

For a certain time interval the Rosenbrock solver always tries to increase time t by the current adaptive step size s and calculates a new vector of quantities $\vec{Q}_{\text{res}}$. After a successful step the events and rules of the model are processed. If this causes a change in $\vec{Q}$, h is first decreased and then the Rosenbrock solver calculates another vector $\vec{Q}_t$ using this adapted step size. The precision of the event processing is therefore determined by the minimum step size $h_{\min}$. The adapt function is defined by Rosenbrock's method [25].

Figure 5 - Calculation of the derivatives at a specific point in time

First, the vector for saving the derivatives of all quantities $\dot{\vec{Q}}$ is set to the null vector $\vec{0}$. Then the rate rules of the model are processed by solving the function $\vec{g}(\vec{Q}, t)$, which can change $\dot{\vec{Q}}$ in some dimensions. After that

for every reaction R_i its velocity ν_i is computed. The derivatives of each species (with index s) participating in the currently processed reaction R_i are updated in each step adding the product of the stoichiometry n_{is} and the reaction's velocity ν_i.

Figure 6 - Architecture of the Systems Biology Simulation Core Library (simplified)

Numerical methods are strictly separated from differential equation systems. The upper part displays the unified type hierarchy of all currently included numerical integration methods. The middle part shows the interfaces defining several special types of the differential equations to be solved by the numerical methods. The class `SBMLinterpreter` (bottom part) implements all of these interfaces with respect to the information content of a given SBML model. Similarly, an implementation of further data formats can be included into the library.

Figure 7 - Simulation of glycolytic oscillations

This figure displays the results of a simulation computed with the Systems Biology Simulation Core Library based on model 206 from BioModels database [7, 29]. Shown are the changes of the concentration (7a) of the most characteristic intracellular metabolites 3-phosphogylcerate, ATP, glucose, glyceraldhyde 3-phosphate, and NAD^+ within yeast cells. Figure 7b displays a selection of the dynamcis of relevant fluxes (D-glucose 6-phosphotransferase, glycerone-phosphate-forming, phosphoglycerate kinase, pyruvate 2-O-phosphotransferase, acetaldehyde forming, ATP biosynthetic process) that were computed as intermediate results by the algorithm. The computation was performed using the Adams-Moulton solver [40] (KiSAO term 280) with 200 integration steps, 10^{-10} as absolute error tolerance and 10^{-5} as relative error tolerance. Due to the importance of feedback regulation the selection of an appropriate numerical solver is of crucial importance for this model. Methods without step-size adaptation, such as the fourth order Runge-Kutta algorithm (KiSAO term 64), might only be able to find a high quality solution with an appropriate number of integration steps. The simulation results obtained by using the algorithm described in this work reproduces the results provided by BioModels database.

Figure 8 - Simulation of the Calvin-Benson cycle

Another example of the capabilities of the Simulation Core Library has been obtained by solving model 390 from BioModels database [7, 30]. This model was simulated using Euler's method (KiSAO term 30) with 200 integration steps for the time interval $[0, 35]$ seconds. This figure shows the evolution of the

concentrations of ribulose 1,5 bisphosphate and the currency metabolites ATP and ADP during the first 35 s of the photosynthesis.

Tables

Table 1 - Comparison of SBML-capable simulators

The table gives an overview about the most characteristic features of SBML-capable simulation programs. As one aspect, it shows, which programs support the most difficult SBML elements (fast reactions, algebraic rules, and events). Another key point is whether all models of the most recent SBML Test Suite can be correctly solved. Please note that in the column for the SBML Test Suite, a dash means that *not all* of its models can be correctly solved, because not all SBML elements are supported. The program iBioSim, for instance, solves the vast majority of the test cases correctly, but does not yet support the delay function, except delayed events. LibSBMLSim, which is a simulation API written in C, can only read models given in SBML Level 2 Version 4 and SBML Level 3. Similarly, a dash in the column for events means that not *all* possible cases for this language element can be correctly solved. COPASI, for instance, supports events in SBML, but not all of the current constructs. Hence, Systems Biology Simulation Core Library and BioUML are the only simulation tools from this selection that pass *all* models of the SBML Test Suite across all levels and versions of SBML.

Program	Version	Difficult SBML elements			Fully SBML Test Suite compliant	SED-ML	Programming language	GUI	API access	Platform	Comments
		Fast reactions	Algebraic rules	Events							
BioUML	0.9.4	✓	✓	✓	✓	In α version	Java	✓	JavaScript	independent	
COPASI	4.8	–	–	(✓)	–	–	C++ (plus multiple bindings)	✓	✓	Windows, Linux, Mac OS X, Solaris	
iBioSim	2.0	✓	✓	✓	✓	–	Java	✓	–	Fedora 15, Windows, Mac OS > 10.6	
JSim	2.07	–	✓	–	–	–	Java	✓	✓	Windows, Linux, Mac OS X	
LibSBMLSim	1.0	✓	✓	✓	L3, L2V4	–	C	–	✓	Windows, Linux, Free BSD, Mac OS X	
VCell	5.0	✓	–	✓	–	–	Java frontend, C/C++ server backend	✓	–	independent	Internet connection required
Simulation Core	1.0	✓	✓	✓	✓	✓	Java	–	✓	independent	

Table 2 - Simulation of the models from the SBML Test Suite (version 2.0.2) using Rosenbrock's solver

The table shows the number of tested models and the total running times of the tests for all SBML levels and versions. An Intel® Core™ i5 CPU with 3.33 GHz and 4 GB RAM was used with Windows® 7 (Version 6.1.7600) as operating system.

Level	Version	Models	Correct simulations	Total running time (in s)
1	2	252	252	1.1
2	1	885	885	3.6
2	2	1,039	1,039	3.3
2	3	1,039	1,039	3.2
2	4	1,041	1,041	3.2
3	1	1,075	1,075	34.3

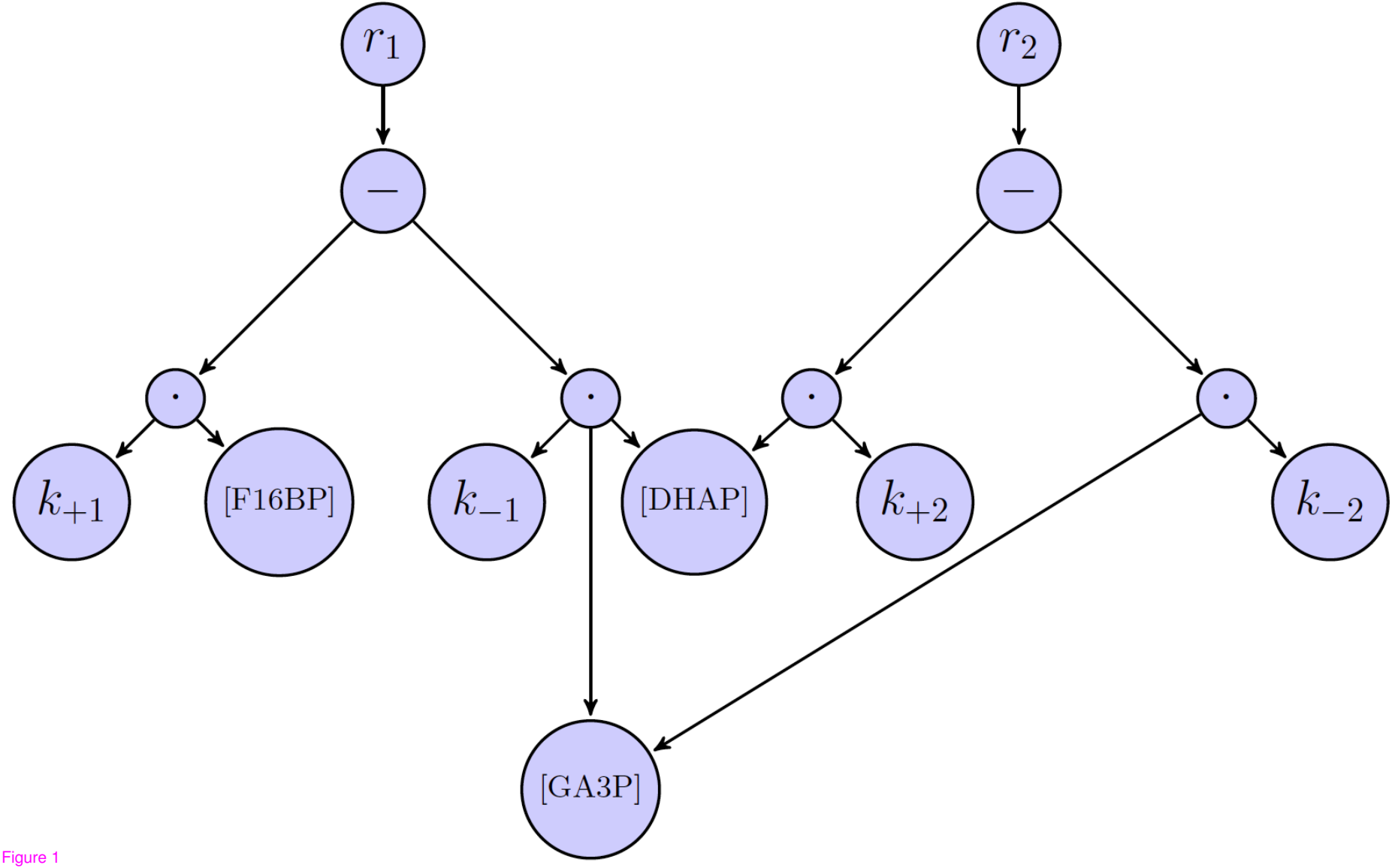


Figure 1

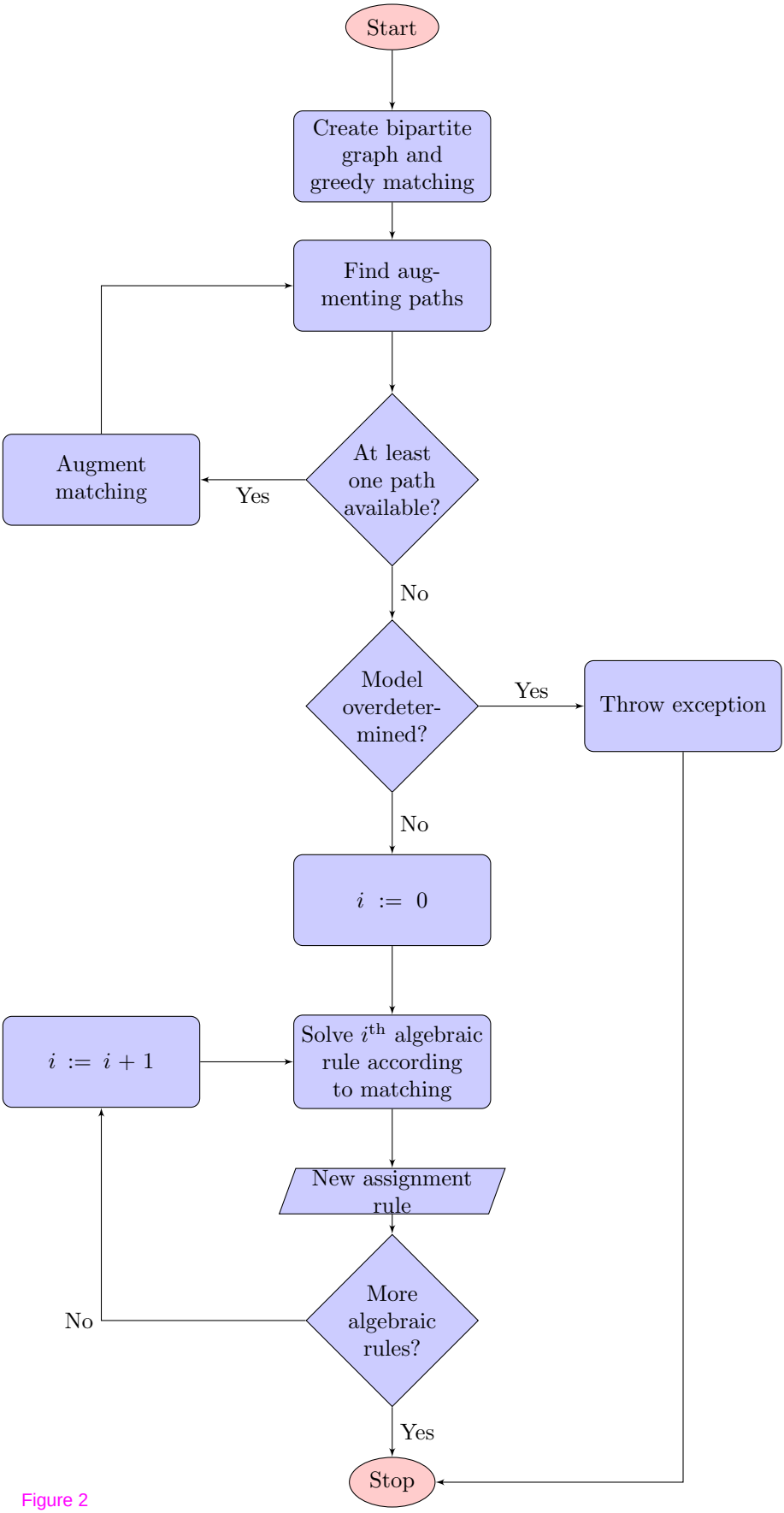

Start
Create bipartite graph and greedy matching
Find augmenting paths
At least one path available?
Yes
Augment matching
No
Model overdetermined?
Yes
Throw exception
No
$i := 0$
Solve i^{th} algebraic rule according to matching
$i := i + 1$
New assignment rule
More algebraic rules?
No
Yes
Stop

Figure 2

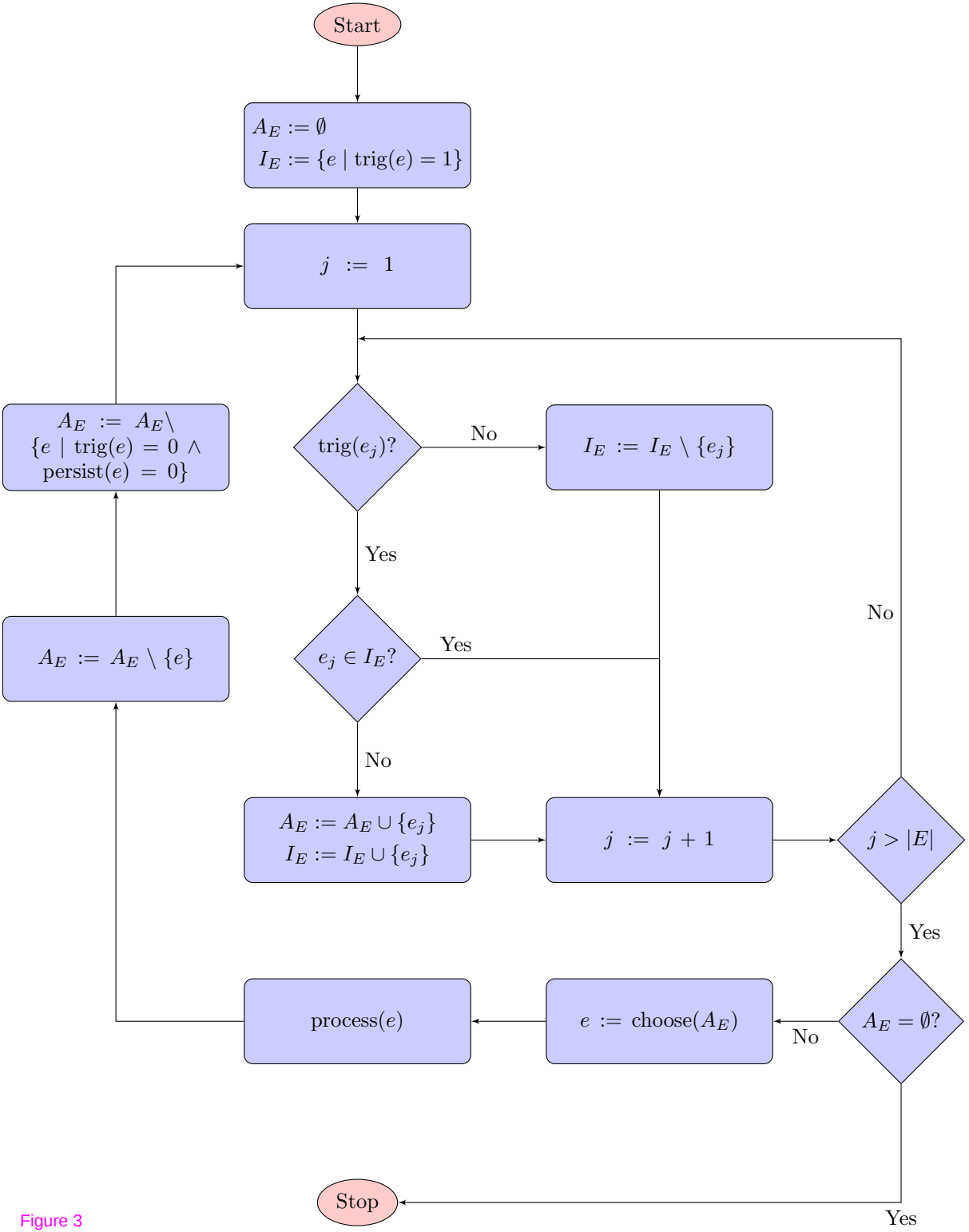

Start
$A_E := \emptyset$
$I_E := \{e \mid \text{trig}(e) = 1\}$
$j := 1$
trig(e_j)?
No
$I_E := I_E \setminus \{e_j\}$
Yes
$e_j \in I_E$?
Yes
No
$A_E := A_E \setminus \{e \mid \text{trig}(e) = 0 \wedge \text{persist}(e) = 0\}$
$A_E := A_E \setminus \{e\}$
$A_E := A_E \cup \{e_j\}$
$I_E := I_E \cup \{e_j\}$
$j := j + 1$
$j > |E|$
No
Yes
$A_E = \emptyset$?
No
$e := \text{choose}(A_E)$
process(e)
Stop
Yes

Figure 3

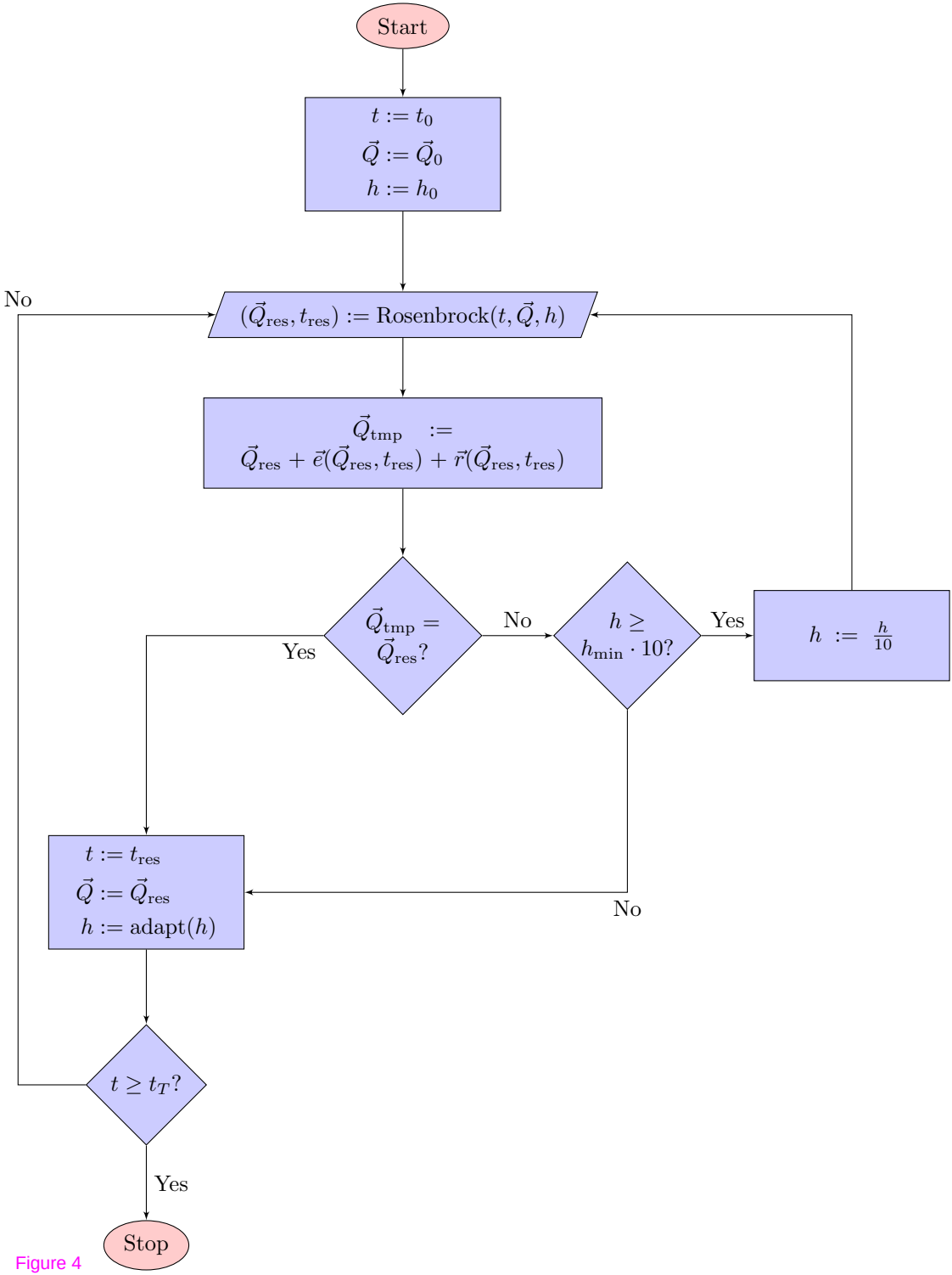

Start
$t := t_0$
$\vec{Q} := \vec{Q}_0$
$h := h_0$
No
$(\vec{Q}_{\text{res}}, t_{\text{res}}) := \text{Rosenbrock}(t, \vec{Q}, h)$
$\vec{Q}_{\text{tmp}} := \vec{Q}_{\text{res}} + \vec{e}(\vec{Q}_{\text{res}}, t_{\text{res}}) + \vec{r}(\vec{Q}_{\text{res}}, t_{\text{res}})$
$\vec{Q}_{\text{tmp}} = \vec{Q}_{\text{res}}$?
Yes
No
$h \geq h_{\text{min}} \cdot 10$?
Yes
$h := \frac{h}{10}$
$t := t_{\text{res}}$
$\vec{Q} := \vec{Q}_{\text{res}}$
$h := \text{adapt}(h)$
No
$t \geq t_T$?
Yes
Stop

Figure 4

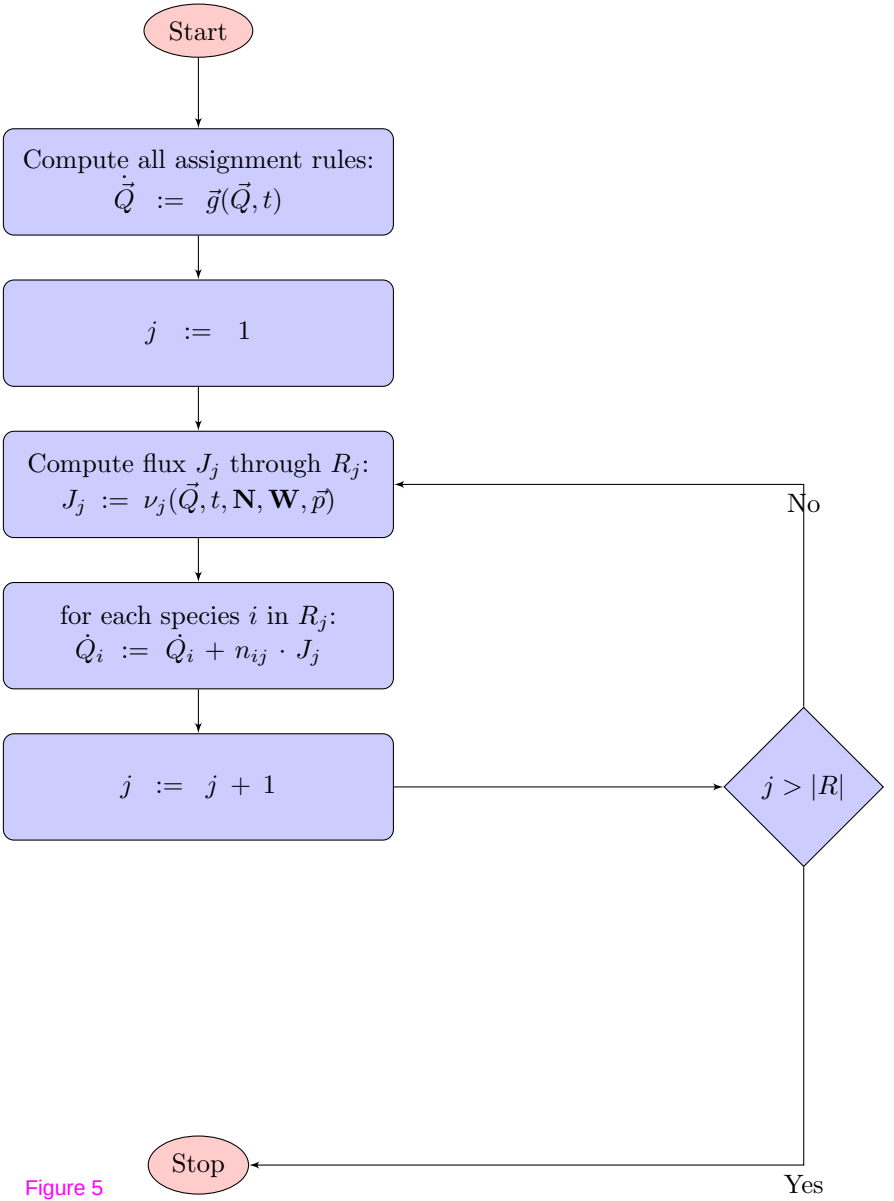
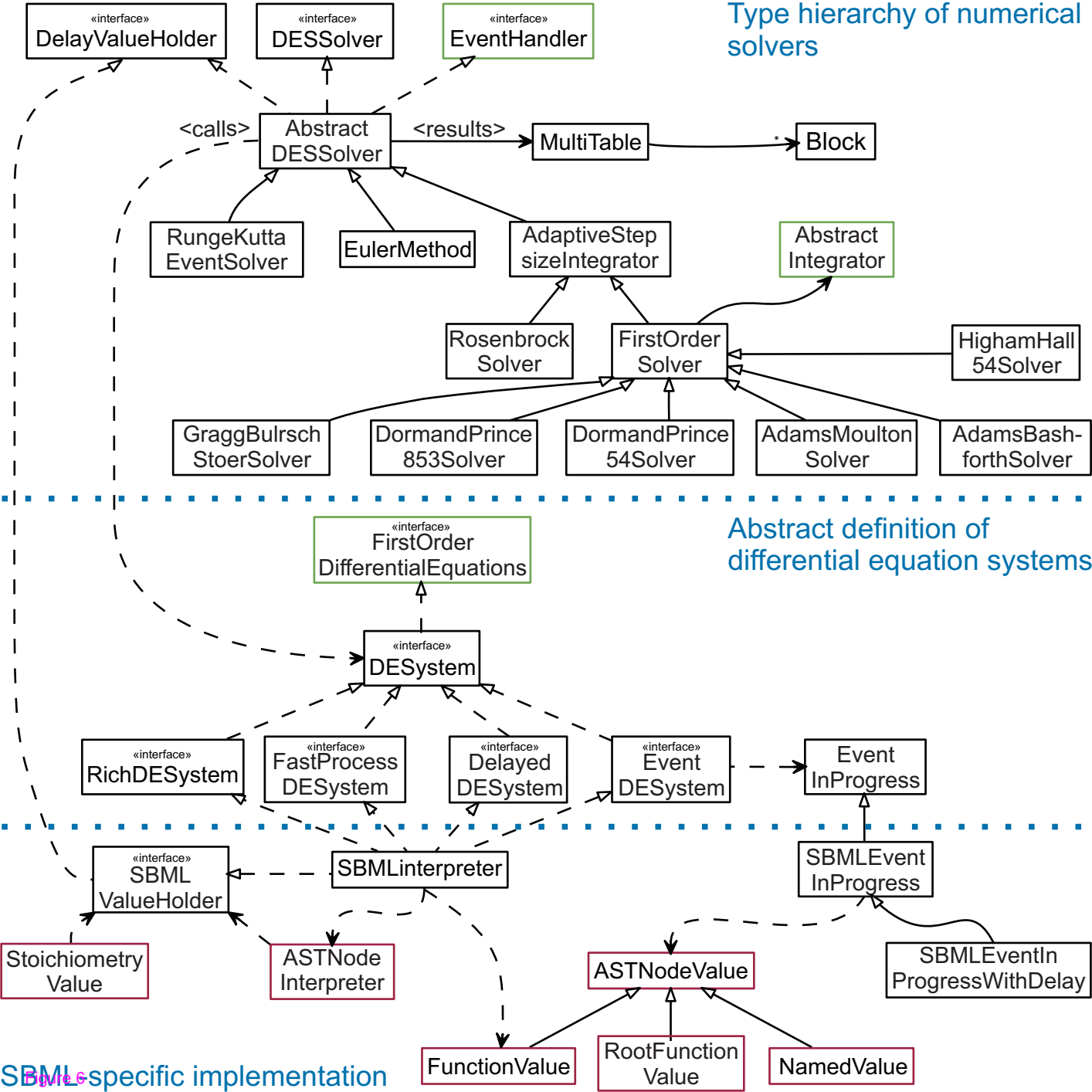

Start
Compute all assignment rules:
$\dot{\vec{Q}} := \vec{g}(\vec{Q}, t)$
$j := 1$
Compute flux J_j through R_j:
$J_j := \nu_j(\vec{Q}, t, \mathbf{N}, \mathbf{W}, \vec{p})$
for each species i in R_j:
$\dot{Q}_i := \dot{Q}_i + n_{ij} \cdot J_j$
$j := j + 1$
$j > |R|$
No
Yes
Stop

Figure 5


«interface»
DelayValueHolder
«interface»
DESSolver
«interface»
EventHandler
Type hierarchy of numerical solvers
<calls>
Abstract DESSolver
<results>
MultiTable
*
Block
RungeKutta EventSolver
EulerMethod
AdaptiveStep sizeIntegrator
Abstract Integrator
Rosenbrock Solver
FirstOrder Solver
HighamHall 54Solver
GraggBulrsch StoerSolver
DormandPrince 853Solver
DormandPrince 54Solver
AdamsMoulton Solver
AdamsBash-forthSolver
«interface»
FirstOrder DifferentialEquations
Abstract definition of differential equation systems
«interface»
DESystem
«interface»
RichDESystem
«interface»
FastProcess DESystem
«interface»
Delayed DESystem
«interface»
Event DESystem
Event InProgress
«interface»
SBML ValueHolder
SBMLinterpreter
SBMLEvent InProgress
Stoichiometry Value
ASTNode Interpreter
ASTNodeValue
SBMLEventIn ProgressWithDelay
FunctionValue
RootFunction Value
NamedValue
SBML-specific implementation
Figure 6

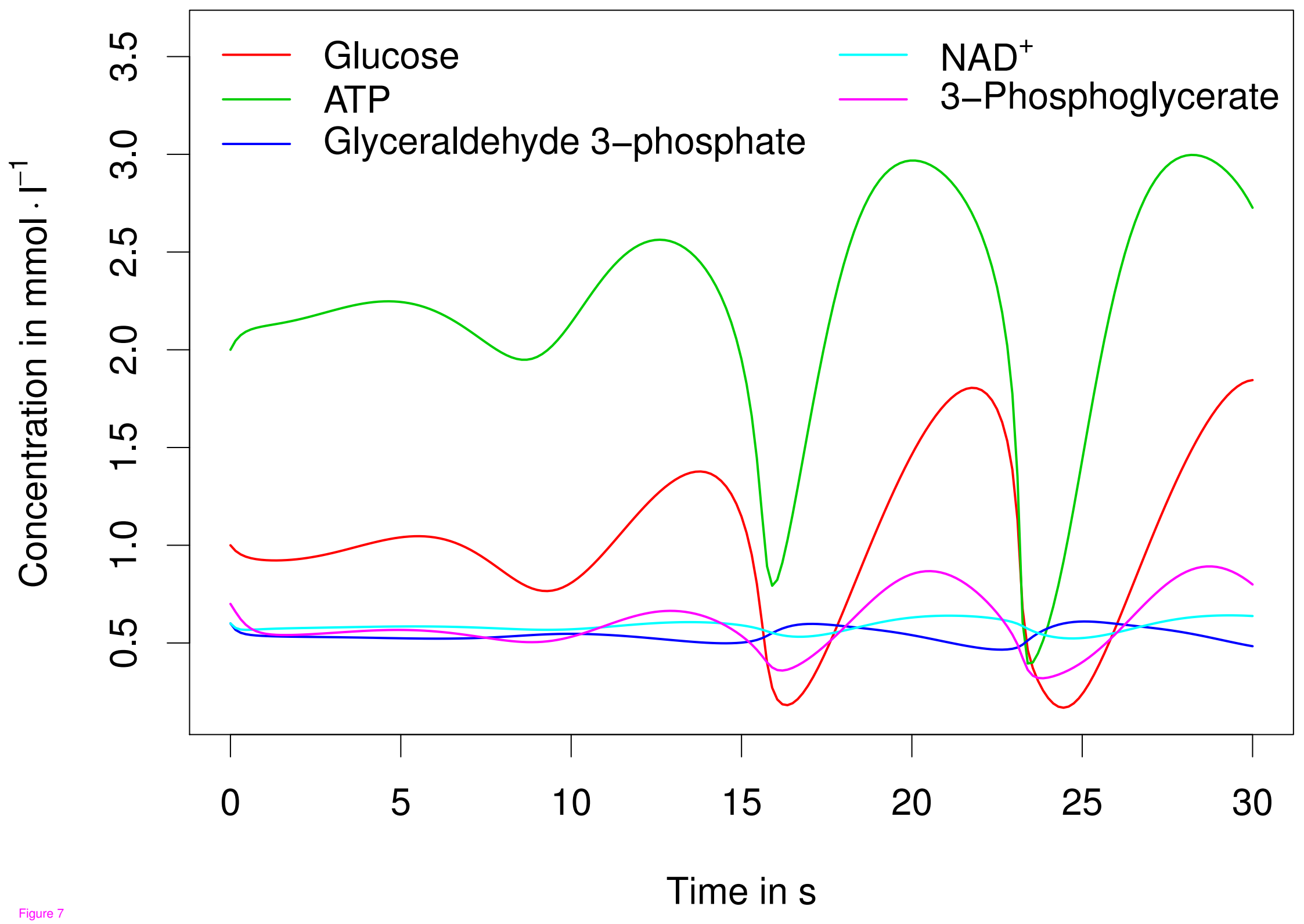

Glucose
ATP
Glyceraldehyde 3–phosphate
NAD+
3–Phosphoglycerate
Concentration in mmol · l−1
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
5
10
15
20
25
30
Time in s

Figure 7

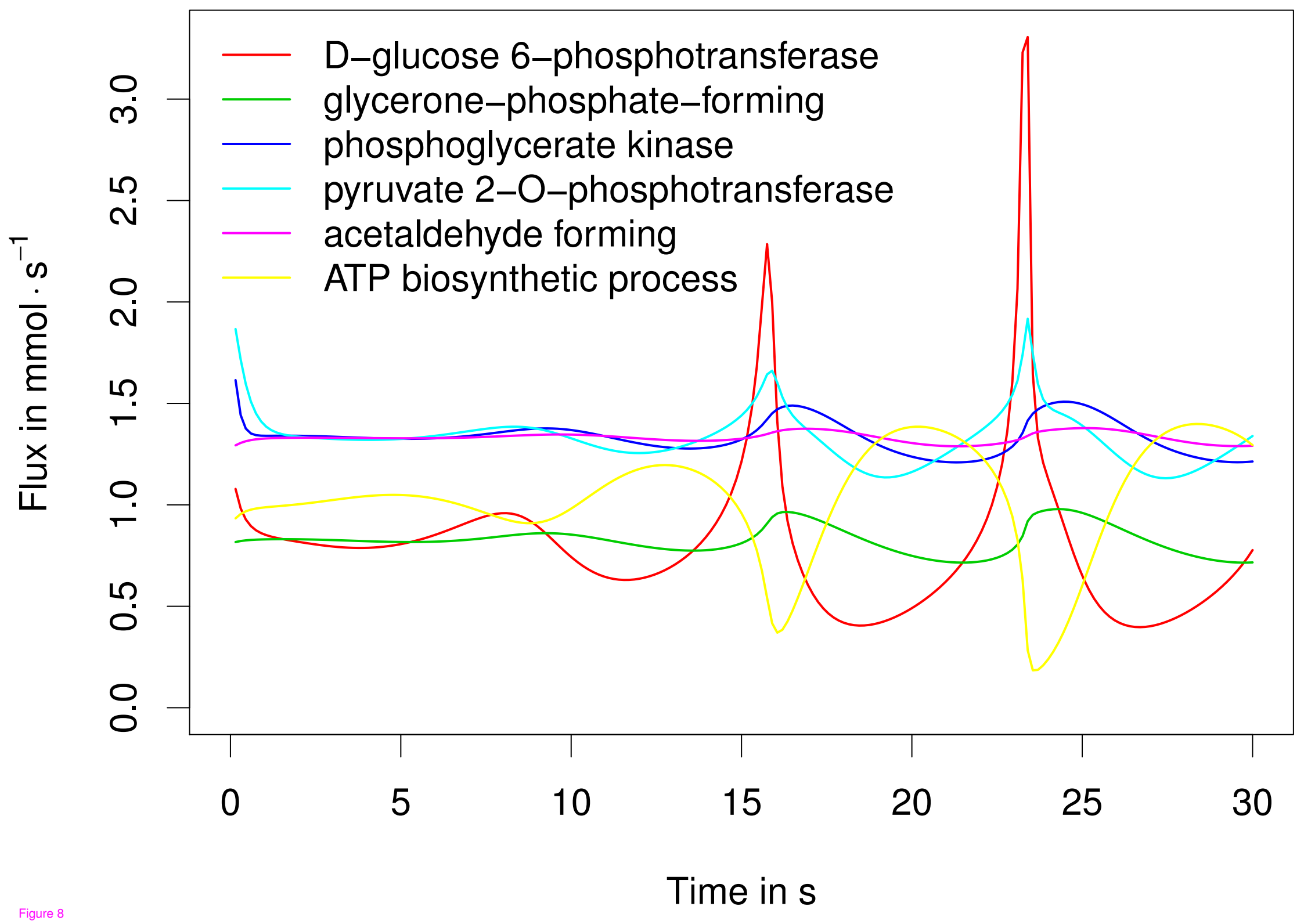

D−glucose 6−phosphotransferase
glycerone−phosphate−forming
phosphoglycerate kinase
pyruvate 2−O−phosphotransferase
acetaldehyde forming
ATP biosynthetic process
Flux in mmol · s−1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
5
10
15
20
25
30
Time in s

Figure 8

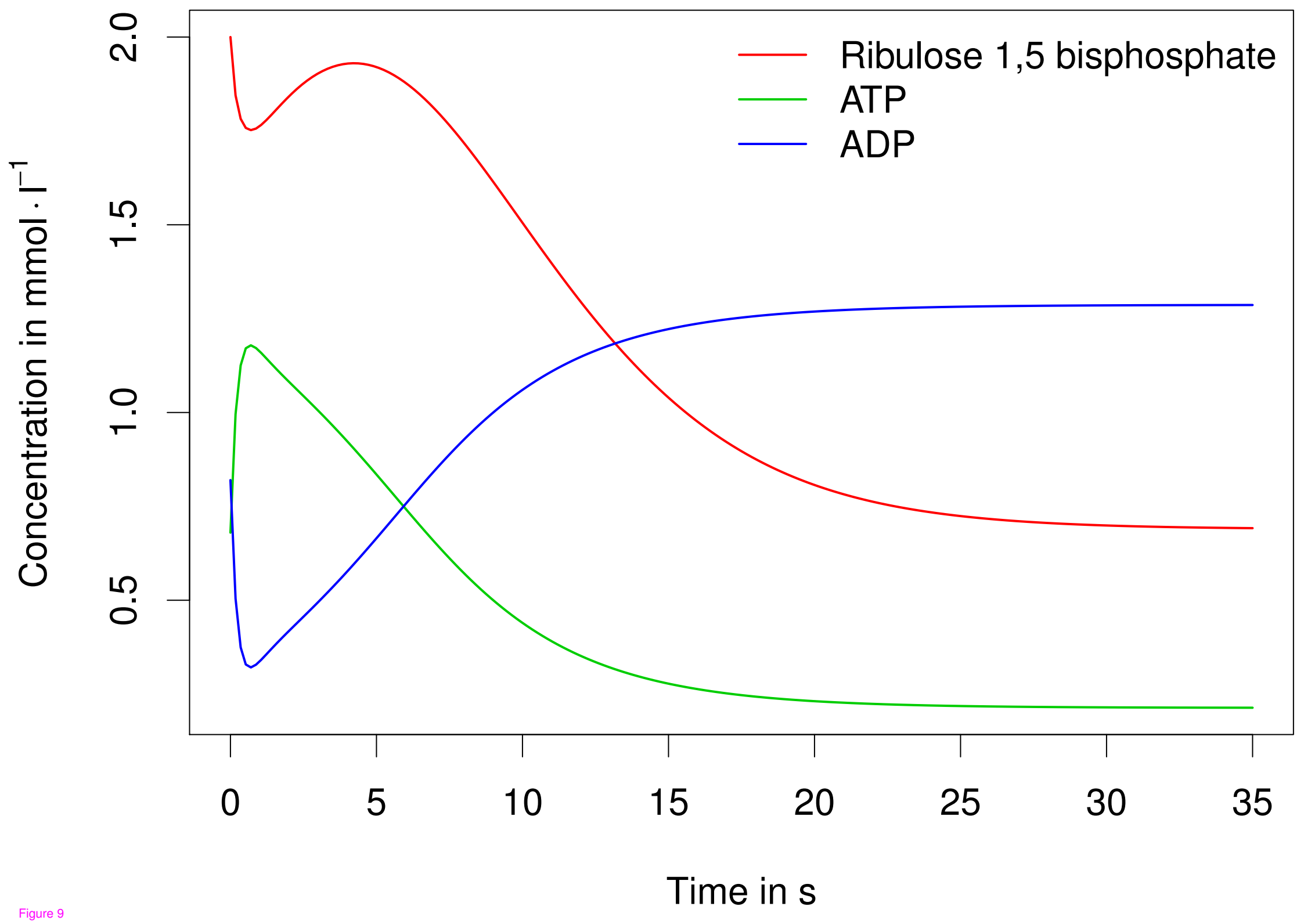

Ribulose 1,5 bisphosphate
ATP
ADP
Concentration in mmol · l⁻¹
0.5
1.0
1.5
2.0
0
5
10
15
20
25
30
35
Time in s

Figure 9