Formal Methods for Systems Biology: Contributions

Elisabetta de Maria

Habilitation à Diriger des Recherches

Formal Methods for Systems Biology: Contributions

Elisabetta De Maria
Université Côte d'Azur, Laboratoire I3S, France

Committee

Alberto Policriti, Universitá degli Studi di Udine, Italy (reviewer)
Anthony Kusalik, University of Saskatchewan, Canada (reviewer)
Ivan Cimrak, University of Zilina, Slovakia (reviewer)
Jonathan Chan, KMUTT University, Thailand (examiner)
Morgan Magnin, École Centrale de Nantes, Laboratoire LS2N, France (examiner)
Jean-Charles Régin, Université Côte d'Azur, Laboratoire I3S, France
Jean-Paul Rigault, Université Côte d'Azur, INRIA SAM, France

June 22, 2020

Thanks

If I could achieve this important step, I have to thank all the members of the project "Constraints and Applications" guided by Jean-Charles Regin, which is part of the team "Discrete Models for Complex Systems" led by Enrico Formenti at I3S research laboratory. I also wish to say thank you to the members of the research team STARS headed by François Bremond at INRIA Sophia Antipolis, where I am conducting a research stay since September 2018. Both at I3S and INRIA, I could find a stimulating scientific environment which was very propitious to my researches.

I would like to express my gratitude to all my students, from bachelor to Ph.D. ones, for the extraordinary inputs they have provided me during the last years. A special thanks goes to Amy Felty and Sabine Moisan for having decided to share with me the adventure of supervising Ph.D. students.

While writing these lines, I am at home because of the lock-down caused by Covid-19. I spend all my days with my son Arthur, who is five years old. He is always happy and passionate, often funny, sometimes joking, and never tired. I smile on him for making my days dense and unforgettable.

Contents

Chapter 1

Curriculum Vitæ with Publication List

1.1 Identification

First name: Elisabetta
Last name: DE MARIA
Date of birth: 15/10/1981
Position: Associate professor (Maître de conférences)
Establishment: Université Côte d'Azur, France
Research laboratory : Laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis (UMR 7271 UNS/CNRS), France.

1.1.1 Professional Background

September 2018 - August 2020. On-leave at INRIA Sophia Antipolis Méditerannée, France, STARS team.

Since September 2011. Associate professor at Université Côte d'Azur, France.

April 2009 - August 2011. Post-doctoral fellow at INRIA Rocquencourt (Paris) under the supervision of François Fages. The first year of my post-doctoral studies was funded by ERCIM (European Research Consortium for Informatics and Mathematics), an organization that annually selects talented researchers for post-doctoral fellowships encouraging mobility. The second year was funded by the international project ERASysBio+ C5Sys on the cell cycle and circadian clock in tumor processes.

1.1.2 Diplomas

March 2009. Ph.D. in Computer Science at University of Udine, Italy. Title of the thesis: "Computer Science Logic for Structure Prediction, String Comparison, and Biological

Pathway Analysis". Mention: excellent. Director: prof. Angelo Montanari. Committee: prof. Angelo Montanari (president), prof. Anthony Jameson (member), and dr. Sebastian Will (secretary).

July 2005. Master degree in Computer Science at University of Udine, Italy. Grade: 110/110 cum laude. Title of the report: "Verifica di schemi di workflow con vincoli temporali mediante automi temporizzati" (Verification of workflow schemes with time constraints using timed automata).

July 2003. Bachelor degree in Computer Science at University of Udine, Italy. Grade: 110/110 cum laude. Title of the report: "Gestione di vincoli temporali in sistemi di workflow" (Management of time constraints in workflow systems).

July 2000. Scientific high school diploma after five years at "Liceo Scientifico Statale G. Marconi", Conegliano (TV), Italy. Grade: 100/100.

1.2 Teaching Activities

In 2019/2010 and 2010/2011, during my post-doctorate at INRIA Rocquencourt (Paris), I taught the following courses:

Fundamentals of Computer Science, tutorial classes, bachelor in Computer Science, 2nd year, University of Versailles-Saint-Quentin-En-Yvelines, France. Course that covers the basics of the C programming language.

Computer Methods for Systemic and Synthetic Biology, lecture course, Master Parisien de Recherche en Informatique (MPRI), France. Course focused on formal methods for bio-informatics.

Since September 1st, 2011 I work for the Department of Computer Science at Université Côte d'Azur, France, for which I taught the following courses:

Introduction to functional programming, lecture course and practical exercises, bachelor in Computer Science, 1st year. Course focused on the Scheme programming language.

Introduction to databases, lecture course and practical exercises, bachelor in Computer Science, 2nd year. Course focused on relational databases.

Compilers, tutorial classes and project, bachelor in Computer Science, 3rd year. Course centered on lexical analysis, syntactic analysis, and code generation.

Model checking, lecture course and tutorial classes, Master de Recherche en Informatique Fondamentale (RIF). Course that covers temporal logics, model checking algorithms, and their complexity.

In addition, my skills in bioinformatics led me to teach the following courses for the Department of Life Sciences at Université Côte d'Azur:

Bio-informatics, project, bachelor Biologie-Informatique-Mathématiques (BIM), 2nd and 3rd year. I supervised some projects based on the statistical analysis of spike trains and some projects on the development of an environment for modeling, simulating, and verifying bio-chemical systems.

Algorithmic for biology, lecture course and tutorial classes, bachelor Biologie-Informatique-Mathématiques (BIM), 2nd and 3rd year. Course that aims at giving an overview of algorithmic problems and highlights applications to biology.

Databases for biology, lecture course and tutorial classes, master Biologie-Informatique-Mathématiques (BIM), 1st year. Course focused on the design and implementation of databases with a biology-related application domain.

Gene regulatory networks, lecture course, master in Computational Biology and Biomedicine (CBB), 2nd year. Course that covers the use of formal methods of computer science to verify bio-chemical systems.

1.3 Publications and Scientific Production (after obtaining my Ph.D.)

Editorial Duties.

- **Elisabetta De Maria**, Ana Fred, and Hugo Gamboa: Proceedings of the 13th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2020) - Volume 3: BIOINFORMATICS, Valletta, Malta, February 24-26, 2020. SciTePress, (2020).

- Ana Cecília Roque, Arkadiusz Tomczyk, **Elisabetta De Maria**, Felix Putze, Roman Moucek, Ana L. N. Fred, Hugo Gamboa: Biomedical Engineering Systems and Technologies - 12th International Joint Conference, BIOSTEC 2019, Prague, Czech Republic, February 22-24, 2019, Revised Selected Papers. Communications in Computer and Information Science 1211, Springer, (2020).

- **Elisabetta De Maria**, Ana Fred, and Hugo Gamboa: Proceedings of the 12th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2019) - Volume 3: BIOINFORMATICS, Prague, Czech Republic, February 22-24, 2019. SciTePress, (2019).

Book.

- **Elisabetta De Maria**: Systems Biology Modelling and Analysis: Formal Bioinformatics Methods and Tools. Contract signed on February 6th, 2020 with the editing house Wiley. Chapters: Introduction, Petri Nets, Boolean Networks, Process Algebras, Rule-based Languages, Pathway Logic, Answer Set Programming, Timed and Hybrid Automata, Ordinary and Stochastic Differential Equations, Conclusion.

Book Chapters.

- **Elisabetta De Maria**, Joëlle Despeyroux, Amy Felty, Pietro Lio, and Carlos Olarte: Computational Logic for Systems Biology, Biomedicine, and Neuroscience. Accepted for publication as a chapter in an "ISTE-Wiley" book, (2020).
- **Elisabetta De Maria**, and Cinzia di Giusto: Inferring the Synaptical Weights of Leaky Integrate and Fire Asynchronous Neural Networks modelled as Timed Automata. Communications in Computer and Information Science 1024, Springer 2019, 149-166, (2019).
- Giovanni Ciatto, **Elisabetta De Maria**, and Cinzia di Giusto: Spiking Neural Networks as Timed Automata. Proc. of the Thematic Research School on Advances in Systems and Synthetic Biology (ASSB), EDP Sciences, (2017).

International Peer-reviewed Journals.

- **Elisabetta De Maria**, Morgan Magnin: Introduction to JBCB Special Issue on CSBio 2019. Journal of Bioinformatics and Computational Biology, World Scientific Publishing Europe Ltd, to appear, (2020). *Impact factor: 0.845.*
- **Elisabetta De Maria**, Cinzia Di Giusto, and Laetitia Laversa: Spiking neural networks modelled as timed automata: with parameter learning. Natural Computing, 1-21, https://doi.org/10.1007/s11047-019-09727-9, (2019). *Impact factor: 1.330.*
- **Elisabetta De Maria**: Introduction to JBCB Special Issue on BIOINFORMATICS 2019. Journal of Bioinformatics and Computational Biology 17 (5), World Scientific Publishing Europe Ltd, (2019). *Impact factor: 0.845.*
- **Elisabetta De Maria**, François Fages, Aurélien Rizk, and Sylvain Soliman: Design, optimization and predictions of a coupled model of the cell cycle, circadian clock, DNA repair system, irinotecan metabolism and exposure control under temporal logic constraints. Theoretical Computer Science, 412(21): 2108-2127, (2011). *Impact factor: 0.718.*

International Disclosure Reviews.

- François Fages, Gregory Batt, **Elisabetta De Maria**, Dragana Jovanovska, Aurélien Rizk, and Sylvain Soliman: Computational systems biology in BIOCHAM. ERCIM News, 82, (2010).

Conference and Workshop Papers with Peer Review.

- **Elisabetta De Maria**, Thibaud L'Yvonnet, Sabine Moisan, and Jean-Paul Rigault: Probabilistic activity recognition for serious games with applications in medicine. Proceedings of the Seventh International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'19), Springer International Publishing, 106-124, (2020). *Selected for publication in the journal Science of Computer Programming.*

- Abdorrahim Bahrami, **Elisabetta De Maria**, and Amy Felty: Modelling and Verifying Dynamic Properties of Biological Neural Networks in Coq. Proceedings of 9th International Conference on Computational Systems Biology and Bioinformatics (CSBIO 2018), ACM, Article No. 12, 1-11, (2018).

- **Elisabetta De Maria** and Cinzia Di Giusto: Parameter Learning for Spiking Neural Networks modelled as Timed Automata. Proceedings of 9th International Conference on Bioinformatics Models, Methods and Algorithms (BIOINFORMATICS 2018), 17-28, (2018). ***Finalist paper*** for the ***Best Paper Award***.

- **Elisabetta De Maria**, Daniel Gaffé, Cédric Girard Riboulleau and Annie Ressouche: A Model-checking Approach to Reduce Spiking Neural Networks. Proceedings of 9th International Conference on Bioinformatics Models, Methods and Algorithms (BIOINFORMATICS 2018), 89-96, (2018).

- **Elisabetta De Maria**, Cinzia Di Giusto, and Giovanni Ciatto: Formal Validation of Neural Networks as Timed Automata. Proceedings of 8th International Conference on Computational Systems Biology and Bioinformatics (CSBIO 2017), ACM, 15-22, (2017).

- **Elisabetta De Maria**, Thibaud L'Yvonnet, Daniel Gaffé, Annie Ressouche, and Franck Grammont: Modelling and Formal Verification of Neuronal Archetypes Coupling. Proceedings of 8th International Conference on Computational Systems Biology and Bioinformatics (CSBIO 2017), ACM, 3-10, (2017).

- **Elisabetta De Maria**, Alexandre Muzy, Daniel Gaffé, Annie Ressouche, and Franck Grammont: Verification of Temporal Properties of Neuronal Archetypes Modeled as Synchronous Reactive Systems. Proceedings of Hybrid Systems Biology (HSB 2016), LNCS 9957: 97-112, (2016).

- **Elisabetta De Maria**, Joëlle Despeyroux, and Amy P. Felty: A Logical Framework for Systems Biology. Proceedings of Formal Methods in Macro-Biology (FMMB 2014), LNCS 8738: 136-155, (2014).

Posters.

- **Elisabetta De Maria**, François Fages Fages, Aurélien Rizk, and Sylvain Soliman: Design, Optimization and Predictions of a Coupled Model of the Cell Cycle, Circadian Clock, DNA Repair System, Irinotecan Metabolism and Exposure Control under Temporal Logic Constraints. Poster at 9th European Conference on Computational Biology (ECCB 2010), (2010).

Research Reports.

- **Elisabetta De Maria**, Alexandre Muzy, Daniel Gaffé, Annie Ressouche, and Franck Grammont: Verification of Temporal Properties of Neuronal Archetypes Using Synchronous Models. INRIA Research Report, (2016).
- **Elisabetta De Maria**, Joëlle Despeyroux, and Amy P. Felty: A Logical Framework for Systems Biology. INRIA Research Report, (2014).

Under Review.

- **Elisabetta De Maria**, Abdorrahim Bahrami, Thibaud L'Yvonnet, Amy Felty, Daniel Gaffé, Annie Ressouche, Franck Grammont: On the Use of Formal Methods to Model and Verify Neuronal Archetypes. Under revision at the journal Frontiers of Computer Science.

1.4 Tools

During my post-doctorate, I contributed to the development of the BIOCHAM software. More specifically, I worked on exporting (resp. importing) Biocham models to (resp. from) other formalisms for modeling biochemical systems (e.g., SBML, ODE, Matlab).
`http://contraintes.inria.fr/BIOCHAM`

1.5 Doctoral and Scientific Supervision

March 2012 - August 2012. *Supervision* of the 2nd year master internship of the student Barghavi Varaprasad, International Research Master CBB (Computational Biology and Biomedicine), Université Nice-Sophia Antipolis. Title of the report: Tests for Gene Regulatory Networks.

June 2014 - September 2014. *Supervision* of the 1st year master internship of the student Steven Roumajon, Master in Computer Science RIF (Recherche en Informatique Fondamentale), Université Nice-Sophia Antipolis. Title of the report: Discrete optimization and temporal logic for neural networks.

June 2016 - December 2016. *Supervision* of the 2nd year master internship of the student Giovanni Ciatto, Laurea Magistrale in Ingegneria Informatica, Universitá di Bologna, Italy. Title of the report: Modeling Third Generation Neural Networks as Timed Automata and verifying their behavior through Temporal Logic.

January 2017 - June 2017. *Supervision* of the 2nd year master internship of the student Thibaud L'Yvonnet, Master in Life Sciences BIM (Biologie-Informatique-Mathématiques), Université Nice-Sophia Antipolis. Title of the report: Modelling and verifying neuronal archetype compositions.

February 2017 - June 2017. *Supervision* of the 1st year master internship of the student Cédric Girard-Riboulleau, Master in Life Sciences BIM (Biologie-Informatique-Mathématiques), Université Nice-Sophia Antipolis. Title of the report: Probabilistic models and verification of neural networks with Prism.

July 2017 - September 2017 *Supervision* of the 1st year master internship of the student Laetitia Laversa, Master in Computer Science RIF (Recherche en Informatique Fondamentale), Université Nice-Sophia Antipolis. Title of the report: Parameter learning for neural networks modelled as timed automata.

Since July 2017. ***Co-direction*** and *supervision* of the Ph.D. student Abdorrahim Bahrami, University of Ottawa, Canada. Title of the Ph.D.: Modelling and verifying dynamical properties of biological neural networks in Coq.

Since December 2018. ***Co-direction*** and *supervision* of the Ph.D. student Thibaud L'Yvonnet, INRIA SAM. Title of the Ph.D.: Relations between human behaviour models and brain models - Application to serious games.

1.6 Dissemination of Scientific Work

June 2009. Research stay of one week and invited talk at Centrum Wiskunde & Informatica (CWI), Amsterdam, Netherlands. Scientific contact: Joke Blom. Title of the talk: Computer Science Logic for Structure Prediction, String Comparison, and Biological Pathway Analysis.

August-September 2009. Presentation of a paper at CMSB2009 (6th International Conference on Computational Methods in Systems Biology), Bologna, Italy. Title of the paper: On Coupling Models Using Model-Checking: Effects of Irinotecan Injections on the Mammalian Cell Cycle.

September 2009. Research stay of one week and invited talk at Fraunhofer Institute for Algorithms and Scientific Computing SCAI, Sankt Augustin, Germany. Scientific contact: Martin Hofmann-Apitius. Title of the talk: On Coupling Models using Model-Checking. Effects of Irinotecan Injections on the Mammalian Cell Cycle.

April 2010. Invited talk at the pole "Modèles Discrets pour les Systèmes Complexes" (MDSC) of Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis (I3S), France. Title of the talk: Computer Science Logic for Structure Prediction, String Comparison, and Model Coupling.

June 2010. Participation at "Ecole Jeunes Chercheurs: Modélisation formelle de réseaux de régulation biologique", Ile de Porquerolles, France.

July 2010. ***Invited speaker*** at the session "Applications of Operational Research in Network & Systems Biology" at the conference EURO2010 (24th European Conference of Operational Research), Lisbon, Portugal. Title of the talk: A Temporal Logic Constraint Solving Approach to Model Coupling. Case Study on the Effects of Irinotecan Injections on the Mammalian Cell Cycle.

September 2010. Presentation of a paper at the French-speaking conference SFC2010 (42ème Congrès annuel de la Societé Francophone de Chronobiologie), La Colle sur Loup, France. Title of the talk: A coupled model of cell cycle, circadian clock, DNA-damage repair mechanism, and Irinotecan metabolism.

September 2010. Presentation of a poster at the international conference ECCB10 (9th European Conference on Computational Biology), Gand, Belgium. Title of the poster: Design, Optimization and Predictions of a Coupled Model of the Cell Cycle, Circadian Clock, DNA Repair System, Irinotecan Metabolism and Exposure Control under Temporal Logic Constraints.

November 2010. Invited talk at "Institut de Recherche en Communications et Cybernétique de Nantes" (IRCCyN), France. Title of the talk: Computer Science Logic for Structure Prediction, String Comparison, and Model Coupling.

February 2011. Invited talk at "Institut de Recherche en Informatique et Systèmes Aléatoires" (IRISA-INRIA), Symbiose team, Rennes, France. Title of the talk: Computer Science Logic for Structure Prediction, String Comparison, and Model Coupling.

March 2011. Invited talk at "Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier" (LIRMM), MAB team, Montpellier, France. Title of the talk: Computer Science Logic for Structure Prediction, String Comparison, and Model Coupling.

February 2012. Presentation of my research topics at Université de Nice-Sophia Antipolis, France, during the visit of a delegation of students from Universitá di Bologna, Italy.

May 2012. Invited talk at Imperial College (United Kingdom), Faculty of Engineering, Department of Electrical and Electronic Engineering. Scientific contact: Erol Gelembe. Title of the talk: On Coupling Models using Temporal Logic Constraints. Effects of Irinotecan Injections on the Mammalian Cell Cycle.

July 2014. Invited talk at "5th Workshop on Logic and Systems Biology" associated to CSL/LICS 2014, "Vienna Summer of Logic", Vienna, Austria. Title of the talk: A logical framework for systems biology. Talk given by my co-author Joëlle Despeyroux.

June 2016. Talk at the annual meeting of the interdisciplinary research axis MTC-NSC (Modélisation Théorique et Computationnelle en Neurosciences et Sciences Cognitives) of Université de Nice Sophia-Antipolis, France. Title of the talk: Verification of Temporal Properties of Neuronal Archetypes Modeled as Synchronous Reactive Systems.

October 2016. Presentation of a paper at the international conference HSB 2016 (Hybrid Systems Biology 2016), Grenoble, France. Title of the paper: Verification of Temporal Properties of Neuronal Archetypes Modeled as Synchronous Reactive Systems.

June 2017. Talk at the national workshop "Workshop C@UCA 2017" (Cerveau, Cognition, Comportement, Collectif, Clinique, Computationnel), Fréjus, France. Title of the talk: Modelling and Formal Verification of Neuronal Archetypes Coupling.

December 2017. Presentation of two papers at the international conference CSBIO 2017 (Computational Systems Biology and Bioinformatics 2017), Nha Trang, Vietnam. Titles of the papers: 1) Formal Validation of Neural Networks as Timed Automata, 2) Modelling and Formal Verification of Neuronal Archetypes Coupling.

January 2018. Presentation of two papers at the international conference BIOINFORMATICS 2018 (Bioinformatics Models, Methods and Algorithms), Madeira, Portugal. Titles of the papers: 1) Parameter Learning for Spiking Neural Networks modelled as Timed Automata, 2) A Model-checking Approach to Reduce Spiking Neural Networks.

February 2019. ***Invited speaker*** at the opening panel of BIOSTEC 2019, Prague, Czech Republic, on the theme "IoT and Biomedical Data: Challenges and Opportunities". BIOSTEC 2019 (12th International Joint Conference on Biomedical Engineering Systems and Technologies) is composed of five co-located conferences: Biodevices 2019, Bioimaging 2019, Bioinformatics 2019, Biosignals 2019, and Healthinf 2019.

June 2019. Invited talk at "School of Electrical Engineering and Computer Science", University of Ottawa, Canada. Title of the talk: Parameter Learning for Spiking Neural Networks Modelled as Timed Automata.

September 2019. Invited talk at "Center of Modeling, Simulation, and Interactions" (MSI), Université Côte d'Azur, France. Title of the talk: Parameter Learning for Spiking Neural Networks Modelled as Timed Automata.

1.7 Ph.D. Committees

September 2014 - December 2017. Member of the Ph.D. follow-up committee of Emna Ben Abdallah, MeForBio team (Formal Methods for Bioinformatics), IRCCyN (Institut de Recherche en Communications et Cybernetique de Nantes), Ecole Centrale

de Nantes, France. Title of the thesis: Analysis of behavioural characteristics and perturbations in biological regulatory networks.

March 2017. ***Examiner for the Ph.D. defense jury*** of Marie Defay Favre, LIMOS (Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes), Université Blaise Pascal, Clermont-Ferrand, France. Title of the thesis: Reconstruction of extended $\chi^{'}$-deterministic Petri Nets from $\chi^{'}$ time series.

June 2019. Member of the Ph D. proposal defence jury of Abdorrahim Bahrami, University of Ottawa, Canada. Title of the thesis: Verifying Dynamic Properties of Neural Networks in Coq.

1.8 Scientific Responsibilities

Autumn 2009. ***Reviewer*** for the special edition of AMAI (Annals of Mathematics and Artificial Intelligence) on "Application of Constraints to Formal Verification and AI", (2009).

Spring 2010. ***Reviewer*** for the workshop CS2Bio 2010 (1st International Workshop on Interactions between Computer Science and Biology), Amsterdam, Netherlands.

May 2014. With two colleagues in computer science and a colleague in biology, I applied (as ***coordinator***) for a call of the interdisciplinary research axis MTC-NSC (Modélisation Théorique et Computationnelle en Neurosciences et Sciences Cognitives) of Université de Nice Sophia-Antipolis, France, to supervise an M1 internship. Project title: Discrete optimization and temporal logic of neural networks. We obtained the funding and the internship took place between June and September 2014.

August 2014. ***Reviewer*** for the conference SAT/CSP-ICTAI-2014 (Special Track on SAT and CSP technologies, 26th IEEE International Conference on Tools with Artificial Intelligence), Limassol, Cyprus.

September 2016. With two colleagues in computer science, two colleagues in electronics, and a colleague in biology, we applied for a call of the Excellence Academy "Réseaux, Information et Société Numérique" of Université Côte d'Azur, France. Title of the project: Modeling, verification, simulation and material implant of artificial neural networks. The project was ranked ***first*** out of 25. Funding: 19000€ for internships, missions and equipment.

Summer 2017. Member of the ***program committee*** of BAI@IJCAI 2017 (workshop on Bioinformatics and Artificial Intelligence), held in conjunction with IJCAI (26th International Joint Conference on Artificial Intelligence), Melbourne, Australia.

December 2017. ***Reviewer*** for the conference POST 2018 (7th International Conference on Principles of Security and Trust), Thessaloniki, Greece.

January 2018. ***Chair*** of the session "Pattern Recognition, Clustering and Classification" at the international conference BIOINFORMATICS 2018, Madeira, Portugal.

Summer 2018. Member of the ***program committee*** of the ICML/IJCAI 2018 Workshop on Computational Biology, Stockholm, Sweden.

Spring 2018 - spring 2019. ***Program chair*** of the international conference BIOINFORMATICS 2019 (10th International Conference on Bioinformatics Models, Methods, and Algorithms), which is part of BIOSTEC 2019 (12th International Joint Conference on Biomedical Engineering Systems and Technologies), Prague, Czech Republic.
Tasks: Drafting of the "Call for papers", definition of the program committee, assignment of the received papers to the program committee members for review, choice of acceptance/reject for border-line papers, definition of the conference program with organisation into thematic sessions, organisation of special sessions, choice of the best papers for publication in a special issue of the JBCB journal, choice of the best paper award and the best poster award, edition of the conference proceedings, participation as invited speaker in the BIOSTEC 2019 introductory panel on the theme "IoT and Biomedical Data: Challenges and Opportunities". *Acceptation rate of full papers at the conference* : 12.5%.

Winter 2018 - winter 2019. ***General chair*** and ***program co-chair*** of the international conference CsBio 2019 (10th International Conference on Computational Systems-Biology and Bioinformatics), Nice, France.
Tasks: drafting of the different funding applications, booking the conference rooms, management of the conference budget, drafting of the "Call for papers", definition of the program co-chair and the program committee, selection of the invited speakers, application for publishing in the "ACM digital library", choice of acceptance/reject for border-line papers, definition of the conference program, choice of the best papers for publication in a special issue of the JBCB journal.

June 2019. Facilitator of the brainstorming of the strategic axis "Humain-Biologie" during the meeting of the I3S Laboratory, Fréjus, France.

Summer 2019. ***Guest editor*** of the JBCB journal (Journal of Bioinformatics and Computational Biology) for the review process of selected papers for the special issue on the conference BIOINFORMATICS 2019.

Spring 2019 - spring 2020. ***Program chair*** of the international conference BIOINFORMATICS 2020 (11th International Conference on Bioinformatics Models, Methods, and Algorithms), which is part of BIOSTEC 2020 (13th International Joint Conference on Biomedical Engineering Systems and Technologies), Valletta, Malta. Same tasks as for Bioinformatics 2019.

Spring 2019. Member of the ***program committee*** of the ICML Workshop on Computational Biology 2019, Long Beach, CA, USA.

Science Administration

June 2013. Member of the organizing committee of the thematic CNRS school "Modélisation formelle de Réseaux de Régulation Biologique", Ile de Porquerolles, France (30 participants).

1.9 Other Activities and Responsibilities

The items in this section are presented in decreasing order of importance.

September 2011 - June 2013. ***Coordinator*** of the ***International Master Program*** Computational Biology and Biomedicine of Université Nice-Sophia Antipolis, France. Tasks: course timetable, redefinition of the study program, international recruitment with telephone interviews, reception of foreign students and assistance with accommodation, organization of juries, presence at all the internship defences, administrative aspects for the award of scholarships, promotion of the master program abroad, etc.

Spring 2017. Member of the selection committee for the associate professor position UNS MCF 389 (Constraint programming, foundations of computer science), Université Nice-Sophia Antipolis, France.

Spring 2014. Member of the selection committee for the associate professor position 27-MCF-0864 Galaxie 4236 (Algorithmic and applications), Université Nice-Sophia Antipolis, France.

May 2018. Examiner for the promotion of Jamil Ahmad from Assistant Professor to Tenured Track Associate Professor at RCMS, NUST, Islamabad, Pakistan.

March 2014 - March 2018. Elected member of the management board of the Science Faculty, Université Nice-Sophia Antipolis, France.

September 2013 - May 2016. Elected member of "Comité Permanent des Ressources Humaines" (CPRH), section 27, Université Nice-Sophia Antipolis, France.

November 2010. With three colleagues, I represented INRIA at "Atrium des métiers" organized by Université Pierre et Marie Curie (UPMC), Paris, France. On this occasion we collected and classified more than 100 curricula vitae of Master level students.

September 2012 - August 2013. As part of the program EMMA (Erasmus Mundus Mobility with Asia), supervisor of the student Sarker Md. Sohel Rana, International Master Program "Computational Biology and Biomedicine", Université Nice-Sophia Antipolis, France.

Chapter 2

Formal Methods for Systems Biology: Contributions

2.1 Introduction

This manuscript is devoted to the researches I performed after obtaining my Ph.D. thesis, in the lapse of time that goes from March 2009 to March 2020. My contributions concern the use of *formal methods* of computer science in the domain of *systems biology*, which is a field that brings together researchers from biological, mathematical, computational, and physical sciences in order to study, conceive, simulate, and make advanced analysis of biological systems [40]. To this aim, biological knowledge is often extracted from high-throughput "omics" (genomics, transcriptomics, proteomics, metabonomics, etc.) data generated thanks to next-generation molecular technologies. The obtained pieces of information are then integrated into *interaction maps* or *networks*, which represent the interactions among the involved biological compounds. In these networks, nodes represent the modelled entities, and edges stand for their interactions. Several kinds of biological networks are in the scope of systems biology: gene regulatory networks (which represent genes, their regulators, and the regulatory relationships between them), protein-protein interaction networks (which model the physical contacts between proteins in a cell), metabolic networks (which represent the biochemical reactions catalyzed by enzymes in a cell), biological neural networks (which describe how neurons in the brain communicate through their synapses), ecological networks (which model the biological interactions of an ecosystem), etc. These networks help in having a *global view on data*, turning data from individual pieces to pieces that connect to form a system.

As stated by Kitano in [46], to understand biology at the system level, we must examine the structure and dynamics of cellular and organismal function, rather than the characteristics of isolated parts of a cell or organism. The core of systems biology actually consists in turning data into networks to which we want to give a *dynamics*. As a matter of fact, these networks are considered as dynamical systems, and we want to study the evolution of their components according to the time evolution. The *time dimension* is thus crucial: it is

relevant to link the topology of these networks to their dynamics [69].

It behoves me to underline that the dynamical systems we treat in systems biology are quite different from the ones studied in physics. The main complication derives from the fact that the large-scale data we get are often incomplete, heterogeneous, and inter-dependent (dependencies are usually hidden). Furthermore, the data in our possession are too few with respect to the search space. In fact, the search space grows exponentially with the number of measured compounds, which gives an explosion on the number of parameters. As a consequence, it is almost impossible to obtain a unique model from a given data set, and a system is generally described by a family of abstract models [81]. The use of formal methods of computer science is then helpful to reason over these families of models and to discriminate models according to the biological properties they satisfy [31].

The fundamental role of formal methods for systems biology will be more extensively discussed in the following sections. In this manuscript, I will show how formal methods greatly helped me in addressing some real biological and medical issues/concerns :

- understand the functioning of some processes which are central in cancer treatments (cell cycle, circadian clock, DNA-damage repair mechanism) in order to optimize time and dose injections of an anti-cancer drug;
- identify some crucial neuronal networks of our brain and study their biological dynamical properties (under several different conditions), which is an important step towards the substantiation of the hypothesis that big neural networks[1] of our brain can be expressed as the composition of some canonical networks, and more generally towards a better understanding of the human brain.

2.2 Why Do We Write Models?

Systems biology seeks at pointing out the emergence of biological phenomena (a system is more than the sum of its parts), and it would be reductive to consider the modelling task as a way to store as more information as possible concerning a biological system. Models are rarely an end in themselves, and they are very often written in order to answer some specific questions. During my last decade of researches, I could identify the following main reasons for writing formal models of biological systems.

Validating/refuting a biological hypothesis. Biologists often get in touch with computer scientists, mathematicians, and/or physicists because they conjecture an hypothesis concerning a biological system and they need help in validating/refuting this hypothesis. In this case, we need to formalize not only the biological system at issue, but also the associated (set of) hypothesis. Tools such as model checkers [15] or theorem provers [5] are then employed to test whether the hypothesis holds or not in the system. For instance, in [24] the authors model a mitogen-activated protein kinase

[1] In neuroscience, the term neuronal is used for small networks made by a few neurons while the term neural is often employed for big networks.

(MAPK) cascade, whose structure corresponds to a sequence of activation of three kinases in the cytosol of a cell, and use model checking techniques to answer to the following question: is the activation of the second kinase of the cascade compulsory for the cascade? In other words: is the presence of the second kinase necessary to produce the output of the cascade? The answer is that the complexes containing this kinase are only needed to regulate the cascade, but they are not mandatory for the signal transduction.

Making predictions. To avoid to perform wet experiments in laboratory, models can be exploited for their predictive power. In this regard, irrespective of whether experiments can be expensive, time consuming, and intrusive for living creatures, we should consider that wet experiments are sometimes unsatisfactory for lack of *operability* (some compounds cannot be manipulated) and *observability* (the expression of some compounds cannot be observed). Models can thus help in predicting the values of some entities that cannot be observed. They can also predict the reactions the system will have under some special conditions, for example when confronted with external factors such as disease, medicine, and environmental changes [76]. For instance, in [51] we consider a coupled model of the mammalian cell cycle and the circadian clock and predict how the cell cycle reacts to circadian gene/protein mutations. The predicted behaviours need to be formally specified to automate reasoning on them.

Associating parameters with biological phenomena. Biologists often expect biological systems to display some known behaviours, but they do not always know under which conditions these behaviours can be observed. To tackle this problem, it is needed to write a model formalizing the biological knowledge concerning the system at issue, and to look for parameters such that the expected dynamical properties are true in the model (these properties are often encoded using formal methods). *Parameter search* is crucial in artificial intelligence, and techniques from this domain are employed in systems biology to infer parameters of biological systems. For instance, in the already cited [51], we search for parameters that allow to fix the exposure times of an anti-cancer drug such that its efficacy on damaged cells is maximized. Observe that parameter search can also be exploited to find parameters such that a given model can reproduce some given execution traces (*data-fitting*). Again, these traces can be formally encoded.

Whatever the reason for writing a model is, formal methods are thus important for having deep insights on the system(s) involved. Of course they can also be exploited to *validate models* w.r.t. some already acquired biological knowledge, i.e., to verify whether the encoded models can display some behaviours they are supposed to show (and do not display some unwanted behaviours). Also in this case the behaviours to be checked should be formally encoded.

In systems biology, models are not static and they are often the object of modifications to incorporate new biological knowledge. Discovery passes through unceasing rounds between

modeling and experiments. Therefore it is like a cycle in which experiments lead to proposition of new improved models and in turn these models suggest enhanced experiments and so on (see Figure 2.1). Formal methods are again necessary to verify the new information is correctly encoded, and consistent with the previous knowledge.

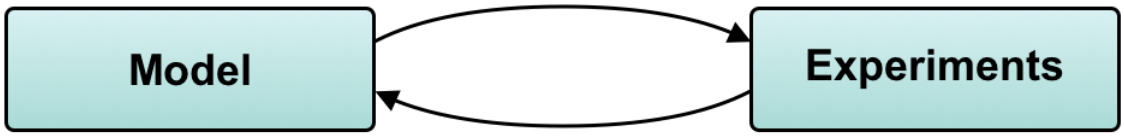


Figure 2.1: Cycle between modeling and experimentation.

To summarize, I would say formal methods are unavoidable to understand and control biological systems, and more generally, to automate reasoning on the different behaviours of biological systems.

2.3 Modelling and Validating Biological Systems: Three Steps

As stated before, formal methods greatly help in understanding how complex biological systems work. Formal approaches for systems biology usually follow the next three steps:

1. describing the system at issue using a formal language;
2. formalizing the biological properties to be verified using a formal language, which is not necessary the language used at step 1);
3. using a tool to (automatically) test whether the encoded properties are verified by the modelled system, or to learn the conditions allowing the property satisfaction.

As far as the first step is concerned, a biological system can be modeled as graph (*transition system*) whose nodes represent the different possible configurations of the system and whose edges encode meaningful configuration changes. Most of the formal languages employed in systems biology allow to (directly or indirectly) represent a biological system as a transition system. These formalisms are discussed in Section 2.4.

Concerning the second step, a biological property concerning the temporal evolution of the biological species involved in the system can be encoded using formal languages such as temporal logics. These formalisms are introduced in Section 2.5. In these first two steps, the quality of interactions between biologists and computer scientists/mathematicians is very important. A gap between different terminologies often arises, and deep discussions are needed to find a common language.

As far as the third step is concerned, tools such as model checkers and theorem provers are largely used to verify that specific properties of a system hold at particular states. These tools are presented in Section 2.6.

2.4 Modeling Biological Systems

As far as the modelling of biological systems is concerned, in the literature we can find both qualitative and quantitative approaches. In Subsection 2.4.1 we introduce qualitative formalisms for systems biology while Subsection 2.4.2 is devoted to quantitative formalisms.

2.4.1 Qualitative Formalisms

To express the *qualitative* nature of dynamics, the most used formalisms are the following ones.

Petri nets. They are directed-bipartite graphs with two different types of nodes: places and transitions. Places represent resources of the system, while transitions correspond to events that can change the state of resources. Thanks to their graph-based structure, Petri nets are a mathematical formalism allowing an intuitive representation of biochemical networks [65, 11]. They are based on synchronous updating techniques. As an example, Figure 2.2 displays a Petri net modelling the influence of the Raf Kinase Inhibitor Protein (RKIP) on the Extracellular signal Regulated Kinase (ERK) signalling pathway [36].

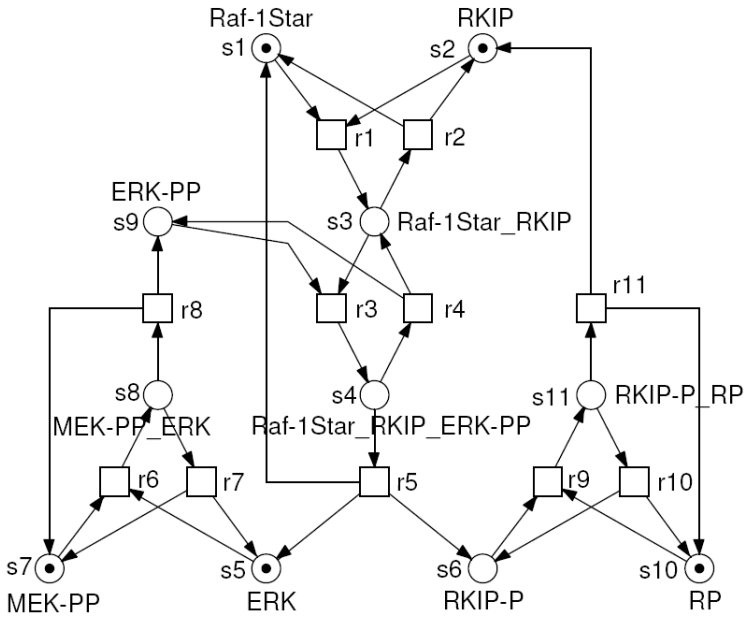


Figure 2.2: Petri net for the core model of the RKIP pathway [36]. It consists of 11 places (represented by circles) and 11 transitions (represented by rectangles).

Boolean networks and Thomas' networks. They are regulatory graphs, where nodes represent regulatory components (e.g,. regulatory genes or proteins) and signed arcs (positive or negative) stand for regulatory interactions (activations or inhibitions). This

graph representation is further associated with logical rules (or logical parameters), which specify how each node is affected by different combinations of regulatory inputs [78]. They can be other synchronous (Boolean networks [44, 71]) or asynchronous (Thomas'networks [77]). An example of Boolean network with a synchronous updating scheme is given in Figure 2.3.

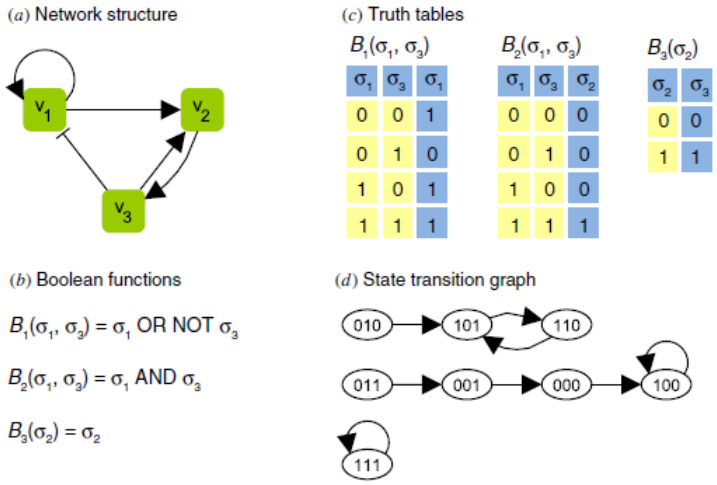


Figure 2.3: A simple Boolean network model [82]. (a) The network structure. The edges with sharp arrows represent positive effects and the edges with blunt arrows represent negative effects. (b) The Boolean functions allowing to compute the values of each variable. (c) The truth tables of the Boolean functions given in (b). (d) The state transition graph of the Boolean model constructed using a synchronous updating scheme.

Reaction rules. Dedicated rule-based languages allow to model biochemical reactions, defining how (sets of) reactants can be transformed into (sets of) products, and associating corresponding rate-laws [10]. The Systems Biology Markup Language (SBML) is the most common representation format for models of biological processes. It is based on XML and it stores models as chemical reaction-like processes that act on entities [39]. The main rule-based modeling tools, namely Biocham [24] and BioNetGen [6], provide both a textual and graphical format and are compatible with SBML. An example of Biocham biochemical reaction rule is the following one: `CycB + CDK => CycB-CDK`, where `CycB` and `CDK` are two proteins and `CycB-CDK` is their complex.

Process algebras. They allow to specify the communication and interactions of concurrent processes without ambiguities. There is a strong correspondence between concurrent systems described by process algebras and biological ones: biological entities may be abstracted as processes that can interact with each other and reactions may be modelled as actions. The most widely used process algebras in systems biology are *pi-calculus* [68], where processes communicate on complementary channels identified by specific names, *bio-ambients* [66], which are based on bounded places where processes are contained and where communications take place, and *process-hitting* [27],

where biological components are abstracted as sorts divided into different processes, interactions between components are represented as a hit from one process of a sort to another process of another sort, and some cooperative sorts allow to represent the combined influences of multiple components on a single target. An intuitive view of *pi-calculus* processes and channels is given in Figure 2.4.

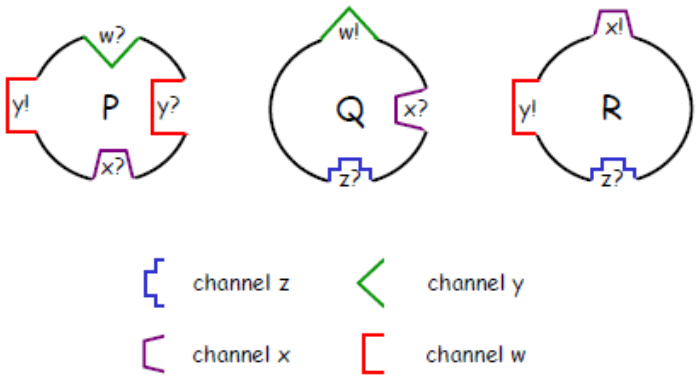


Figure 2.4: *pi*-calculus and channels: an intuitive view [67]. There are three processes, P, Q, and R (ovals), with four communication channels (inputs and outputs have complementary shapes).

Languages for reactive systems. These languages allow to model reactive systems, that is, systems that constantly interact with the environment and which may have an infinite duration. Since many biological systems can be seen as reactive systems continuously reacting to some stimuli, languages for reactive systems such as Nusmv [33] or Lustre [54] are suited to model them. For instance, let me consider the following Lustre equation modelling the potential value of a single input neuron: `p = if pre(Spike) then w else pre(p)+w`. It says that the current potential value `p` only takes into account the weight `w` of the input edge if a spike was emitted at the previous time unit (`pre(Spike)`), while it also takes into account the previous time-unit potential value `pre(p)` if no spike was registered at the previous time unit.

Pure logics. Different extensions of Linear Logic, such as HyLL [20] and SELL [61], allow to specify systems that exhibit modalities such as temporal or spatial ones. In these logics, propositions are called resources, and rules can be viewed as rewrite rules from a set of resources into another set of resources, where a set of resources describes a state of the system. Thus, biological systems can be modeled by a set of rules of the above form. Higher-order logic (HOL [64]) can be also conveniently exploited to formalize reaction kinetics. Pathway Logic uses rewrite theories to formalize biological entities and processes [75]. As an example of rule in logics, the Hyll rule `active(a,b)` $\stackrel{\text{def}}{=}$ `pres(a)` $\rightarrow$ `(pres(a)` $\otimes$ `pres(b))` describes the fact that a state where `a` is present can evolve in one step into a state where both `a` and `b` are present.

Logic programming languages. Declarative problem solving languages belonging to the family of logic programming languages, such as Answer Set Programming (ASP), allow to model biological systems with inherent tolerance of incomplete knowledge, and to generate hypotheses about required expansions of biological models [29, 80, 25]. As an example, the ASP activation rule `act(Y, T+1)` ← `act(X, T), activates(X, Y, T)` says that protein `Y` will be active at time step `T+1` if protein `X` is active and there is an activating connection between `X` and `Y` at the previous time step.

During my post-doctoral researches, I had the chance to use a number of qualitative formalisms to model biological systems. I exploited the rule-based language available in Biocham [24], even contributing to develop the software in order export/import Biocham models to/from other formalisms for modelling biochemical systems (e.g., SBML, ODE, Matlab), the synchronous language for reactive systems Lustre [34], the probabilistic language for reactive systems PRISM [47], the Hybrid Linear Logic Hyll [20], and the language Coq [5], which implements an expressive higher-order logic.

2.4.2 Quantitative Formalisms

To capture the dynamics of a biological system from a *quantitative* point of view, the most used approaches are the following ones.

Ordinary or stochastic differential equations. The most classical quantitative models resort to ordinary differential equations. The interaction between components is captured by sigmoid expressions embedded in differential equations. Both positive and negative regulations can be considered. Models based on ordinary differential equations can hardly be studied analytically, but are often employed to simulate and predict the answer of a biological systems.

To track not only individuals but total populations, stochastic differential equations are often used [74]. A typical stochastic differential equation is of the following form: $dX = b(t, X_t)dt + v(t, X_t)dW_t$, where X is a system variable, b is a Riemann integrable function, v is an integrable function, and W is a continuous-time stochastic process (more precisely, a Brownian Motion). As an example, the growth of tumors under immune surveillance and chemotherapy can be studied using the following stochastic differential equation [43]: $\frac{dx}{dt} = r_0 x(1 - \frac{x}{K}) - \frac{\beta x^2}{1+x^2} + x(1 - \frac{x}{K})A_0 cos(\omega t) + x(1 - \frac{x}{K})W_t$, where x is the amount of tumor cells, $A_0 cos(\omega t)$ denotes the influence of a periodic chemotherapy treatment, r_0 is the linear per capita birth rate of cancer cells, K is the carrying capacity of the environment, and β represents the influence of immune cells.

Hybrid Petri nets. They are characterized by the presence of two kinds of places (discrete and continuous) and two kinds of transitions (discrete and continuous) [37]. A continuous place can hold a non-negative real number as its content, and a continuous transition fires continuously at the speed of an assigned parameter. Biological pathways can be observed as hybrid systems, showing both discrete and continuous evolution. For instance, protein concentration dynamics behave continuously when coupled with

discrete switches. An example of Hybrid Petri net modeling the lac operon gene regulatory mechanism in the bacterium *Escherichia coli* is graphically depicted in Figure 2.5.

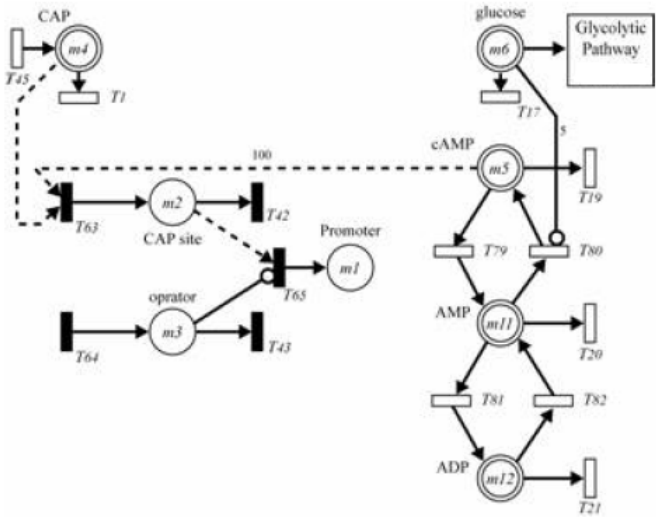


Figure 2.5: Hybrid Petri net for the lac operon gene regulatory mechanism in *Escherichia coli* [57]. Filled rectangles stand for discrete transitions, empty rectangles for continuous transitions, simple circles for discrete places, and double circles for continuous places. Dashed edges represent test arcs, which do not consume any content of the source place of the arc by firing.

Timed automata and hybrid automata. Timed automata [50] are finite state automata extended with timed behaviours: constraints allow to limit the amount of time spent within particular states and the time intervals in which transitions are enabled. They are suited to model time aspects of biological systems, such as time durations of activities. Hybrid automata [9] combine finite state automata with continuously evolving variables. A hybrid automaton exhibits two kinds of state changes: discrete jump transitions occur instantaneously, and continuous flow transitions occur when time elapses. The presence of both discrete and continuous dynamics makes this formalism appealing to model biological systems [8]. In Figure 2.6 I report a hybrid automaton modelling the *Escherichia coli* bacterium.

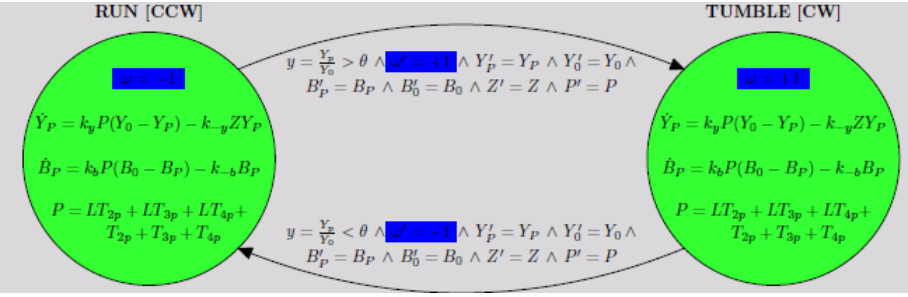
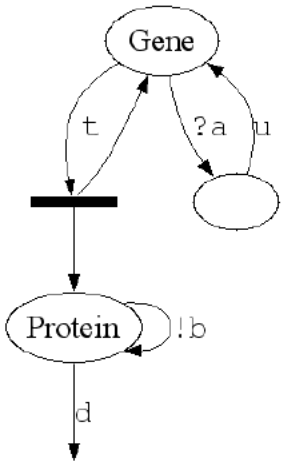


Figure 2.6: Hybrid automaton modelling *Escherichia coli*, a bacterium detecting the food concentration through a set of receptors. It responds in two ways: RUNS (it moves in a straight line by moving its flagella counterclockwise [CCW]) and TUMBLES (it randomly changes its headings by moving its flagella clockwise [CW]). ω is the angular velocity, taking discrete values +1 for CW and -1 for CCW.

Stochastic process algebras. They are process algebras where models are decorated with quantitative information used to generate stochastic processes. The most exploited formalisms are stochastic *pi*-calculus [63], where each channel is associated with a stochastic rate, and process algebras such as Bio-PEPA [14], which allows the specification of complex kinetic formulae. As an example, Figure 2.7 represents a stochastic *pi*-calculus model of a gene with inhibitory control.



$$Gene(a,b) = \tau_t.(Gene(a,b) \mid Protein(b)) + ?a.\tau_u.Gene(a,b)$$

$$Protein(b) = !b.Protein(b) + \tau_d$$

Figure 2.7: A stochastic *pi*-calculus model of a gene with inhibitory control [7]. The gene can transcribe a protein by first doing a stochastic delay at rate t and then executing a new protein in parallel with the gene. Alternatively, it can block by doing an input on its promoter region a, and then unblock by doing a stochastic delay at rate u. The transcribed protein can repeatedly do an output on the promoter region b, or it can decay at rate d. The gene is parameterised by its promoter region a, together with the promoter region b that is recognised by its transcribed proteins.

Rule-based languages with continuous/stochastic dynamics. They allow to write mechanistic models of complex reaction systems, associating continuous or stochastic dynamics to rules [45]. In the popular tool Kappa [19], entities are graphical structures, rules are graph-rewrite directives, and rules fire stochastically, as determined by standard continuous-time Monte Carlo algorithms. As an example, Figure 2.8 illustrates a basic kinase-phosphatase model consisting of three Kappa rules. There are three agents: a Kinase, a Target, and a Phosphatase. A phosphorylation event is simply described by three elementary actions (binding, modification, and unbinding) and their corresponding rules.

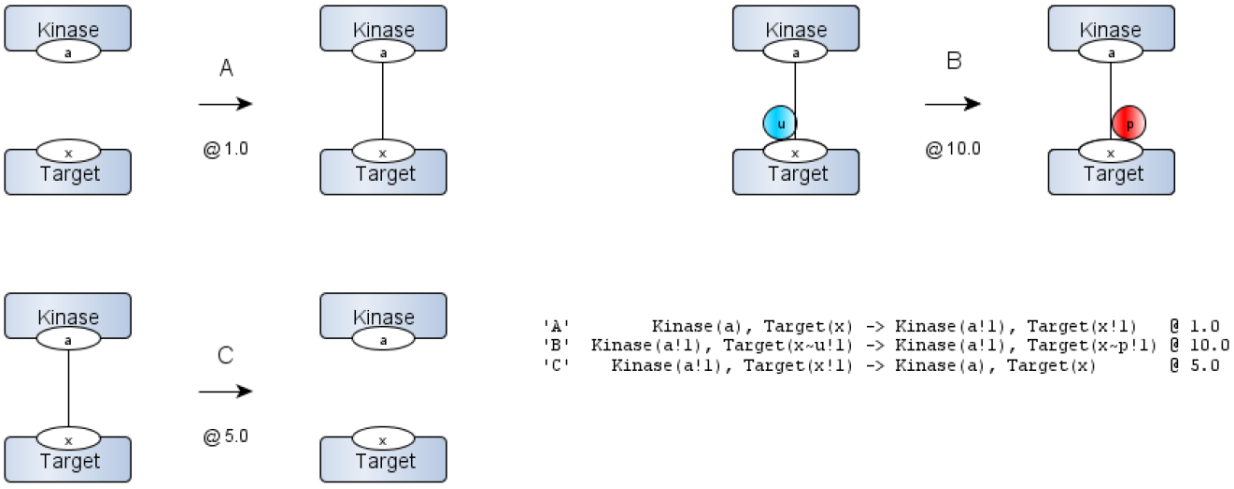


Figure 2.8: The three rules describing a phosphorylation in the kinase-phosphatase model [83]. A) The Kinase binds its Target at site `x`; B) the Kinase phosphorylates the site at which it is bound; C) the Kinase dissociates (unbinds) from its target. In the textual notation, internal states are represented as ∼`u` (unphosphorylated) and ∼`p` (phosphorylated), and bindings as `!` with shared indices across agents indicating the two endpoints of a link. Every rule is associated with a rate constant, which controls the probability of the rule firing during the simulation.

Among these quantitative formalisms, during my post-doctoral researches I have exploited timed automata (encoded with the tool Uppaal [4]) and the Biocham language with continous dynamics [23].

As stated before, systems encoded with the above qualitative and quantitative formalisms can be directly seen as transition systems, or explicitly converted into transition systems. For instance, the Boolean semantics of Biocham associates to a reaction rule model a transition system where the set of states is the set of all tuples of Boolean values denoting the presence or absence of the different biochemical compounds, and the transition relation is the union (i.e., disjunction) of the relations associated to the reaction rules.

Before concluding this section, it is important to underline that a new important trend of the last years consists in automatically inferring the topology of biological systems from

time series data. In [1], the authors provide a logical approach to infer biological regulatory networks based on given time series data and known influences among genes. In [56], the authors propose a statistical learning algorithm to learn both the structure and the reaction rates of chemical reaction networks.

2.5 Specifying Biological Systems

The most widespread formal languages to specify properties concerning the dynamical evolution of biological systems are *temporal logics*. They are formalisms for describing sequences of transitions between states [21].

The *Computation Tree Logic* CTL* [15] allows one to describe properties of computation trees. Its formulas are obtained by (repeatedly) applying Boolean connectives, *path quantifiers*, and *state quantifiers* to atomic formulas. The path quantifier **A** (resp., **E**) can be used to state that all paths (resp., some path) starting from a given state have some property. The state quantifiers are the next time operator **X**, which can be used to impose that a property holds at the next state of a path, the operator **F** (sometimes in the future), that requires that a property holds at some state on the path, the operator **G** (always in the future), that specifies that a property is true at every state on the path, and the until binary operator **U**, which holds if there is a state on the path where the second of its argument properties holds and, at every preceding state on the path, the first of its two argument properties holds.

The *Branching Time Logic* CTL [16] is a fragment of CTL* that allows quantification over the paths starting from a given state. Unlike CTL*, it constrains every state quantifier to be immediately preceded by a path quantifier. The *Linear Time Logic* LTL [72] is another known fragment of CTL* where one may only describe events along a single computation path. Its formulas are of the form $\mathbf{A}\varphi$, where φ does not contain path quantifiers, but it allows the nesting of state quantifiers. The *Probabilistic Computation Tree Logic* PCTL [35] quantifies the different paths by replacing the **E** and **A** modalities of CTL by probabilities.

During my post-doctoral researches, to analyze in deep biological systems, I exploited the logics CTL, LTL, PCTL, and some of their extensions, such as *Constraint-LTL* [22], which enriches LTL with arithmetic constraints.

Other formalisms, such as languages for reactive systems or logics, provide a unified framework to encode not only biological systems but also temporal properties of their dynamic behaviour. For instance, the language Lustre [34] offers an original means to express properties as *observers*. An observer of a property is a program, taking as inputs the inputs/outputs of the program under verification, and deciding at each instant whether the property is violated or not.

In the last years, I could encode dynamical properties of biological systems using the language Lustre [34], the modal linear logic Hyll [20], and the higher-order logic implemented in the Coq theorem prover [5].

I will discuss later the advantages/drawbacks of temporal logics versus logics. At this step, I can say that properties encoded in temporal logics (or equivalent formalisms) can be automatically proved thanks to the use of model checkers. On the other hand, properties

formalized in pure logics can be proved thanks to theorem provers but an expert is needed and the proofs can be time consuming. However, model checking suffers from the state space explosion problem, which sometimes prevents it from terminating in a reasonable amount of time unless specific parameters are chosen. Theorem provers allow to prove more general properties, without having to fill in some of the universal quantifiers with specific values for parameters.

2.6 Validating Biological Systems

Until the eighties, the most common validation techniques for software and hardware systems were simulation and testing, which consist in injecting some signals in a given system at some given times, let the system evolve, and observe the output signals at some given times. The problem is that controlling all the possible interactions and all the bugs of a system using these techniques is rarely possible (too many executions should be considered).

Formal verification was initially introduced in the eighties to prove that a piece of software or hardware is free of errors [15] with respect to a given model. The field of systems biology is a more recent application area for formal verification, and the most common approaches to the formal verification of biological systems are *model checking* [15] and *theorem proving* [5].

In order to apply model checking, the biological system at issue should be encoded as a finite transition system and relevant system properties should be specified using temporal logic. Formally, a transition system over a set AP of atomic propositions is a tuple $M = (Q, T, L)$, where Q is a finite set of states, $T \subseteq Q \times Q$ is a total transition relation (that is, for every state $q \in Q$ there is a state $q' \in Q$ such that $T(q, q')$), and $L : Q \to 2^{AP}$ is a labeling function that maps every state into the set of atomic propositions that hold at that state. Given a transition system $M = (Q, T, L)$, a state $q \in Q$, and a temporal logic formula φ expressing some desirable property of the system, the *model checking problem* consists of establishing whether φ holds at q or not, namely, whether $M, q \models \varphi$. Another formulation of the model checking problem consists of finding all the states $q \in Q$ such that $M, q \models \varphi$. Observe that the second formulation is more general than the first one. There exist several tools to automatically check whether a finite transition system verifies a given CTL, LTL, or PCTL formula, e.g., NuSMV [13], SPIN [38], and PRISM [47]. Probabilistic model checkers such as PRISM allow to directly compute the probabilities for some given temporal logic formulae to hold in the system (provided that some probabilities are associated to the modeled transitions).

Theorem provers are formal proof systems providing a formal language to write mathematical definitions, executable algorithms and theorems, together with an environment for the development of machine-checked proofs. Formulas are proved in a logical calculus and some tactics can be exploited to perform backward reasoning and replace the formula to be proved (conclusion or goal) with the formulas that are needed to prove it (premises or subgoals). To be able to prove properties of biological systems using theorem provers, both systems and specifications should be encoded in the logic implemented by the theorem prover.

State-of-the art theorem provers are Coq [5] and Isabelle/HOL [60].

Let me compare more in detail model checking and theorem proving. The main strength of model checking relies on the fact that it is a press-button methodology. The user just has to encode the system and the temporal logic formula he wants to test, and to press a button to query the tool. Model checkers give a Boolean answer (the formula holds or not in the system), and the user does not have to care about the proof (the proof is transparent to the user). On the other hand, when proofs are involved, theorem provers often do not make them automatically (recent advances in both proof theory and systems often provide us with a partial automation of the proofs).

When a property is not satisfied, model checkers provide us with a counter-example, that is, an execution trace showing the falsification of the property. This trace can give some insights about modifications to do in the system to make the property hold. In a symmetric way, let us suppose theorem provers allow us to prove a property of the system which is not desirable. In this case the proof we get can help us in understanding what should be modified in the system so that the property is not satisfied. More precisely, we can look for the rules to be removed/modified among those that have been used in the proof. Furthermore, a successful proof of a given property can be exploited to prove similar properties with theorem provers.

A strength of theorem proving with respect to model checking is that, when we prove an existential property using certain rules of a model, we have the guarantee that all the models containing such rules satisfy the property. This is important because in biology we often deal with incomplete information. It is also worth noting that in model checking, all objects must be finite: both the number of states, and the number of transitions in the transition system. In logics, objects can potentially be infinite; in particular, we can have an infinite number of states. Thus, although being time consuming and requiring the presence of humans to complete proofs, theorem provers often allow to prove properties with a higher level of generality.

2.7 Contributions

This section describes my main contributions concerning the application of formal methods in systems biology. A subsection is devoted to each selected work. While the first contribution deals with metabolic networks, the other ones focus on biological neural networks (the word "biological" is important because these networks are supposed to be much closer to the brain functioning than the networks used in artificial intelligence). For each contribution, I highlight how formal methods play an important role.

2.7.1 A Coupled Model of the Cell Cycle, Circadian Clock, DNA Repair System, Irinotecan Metabolism and Exposure Control

My first selected contribution is:

Elisabetta De Maria, François Fages, Aurélien Rizk, Sylvain Soliman. *Design, optimization and predictions of a coupled model of the cell cycle, circadian clock, DNA repair system, irinotecan metabolism and exposure control under temporal logic constraints.* Theoretical Computer Science 412(21): 2108-2127 (2011).

The complete paper appears in Chapter 4. In this work we address the coupling of some models that play a role in cancer therapies. Recent advances in cancer *chronotherapy* techniques support the evidence that there exist important links between the cell cycle regulation and the circadian clock genes. Modeling these links is crucial to better understand how to efficiently target malignant cells depending on the phase of the day and patient characteristics. We propose an approach to the investigation of the influence of an anti-cancerogenic drug (irinotecan) on the mammalian cell cycle which is motivated by the following considerations: (i) There are some phases of the cell cycle when irinotecan is more toxic and other phases where it produces less DNA damage. The most toxic phase is synthesis. (ii) Healthy cells are untrained by the circadian clock while cancer cells are often not synchronized. The key idea is thus to inject irinotecan when it is less toxic for healthy cells. In this way, healthy cells should not be sensibly damaged but some tumor cells that happen to be phase shifted could be killed. This concept is at the heart of a collaboration with the oncologist Francis Lévi, CNRS-INSERM, Hopital Paul Brousse, Villejuif, France.

To study the impact of the drug on the cell life, we consider the following four models.

Irinotecan (Ballesta et al. 2011, [2]). This drug is extracted from the cortex and the leaves of a Chinese tree, called the happiness tree. The model describes how irinotecan injections cause DNA damage in presence of an enzyme called Topoisomerase 1.

p53-based DNA-damage repair network (Ciliberto et al. 2005, [12]). p53 is a tumor-suppressor protein that is normally present in cell nuclei in small quantities and whose concentration increases when DNA damage occurs. The idea is that, in normal conditions, protein Mdm2 controls that the concentration of protein p53 in the cell nucleus is feeble. DNA damage increases the degradation rate of protein Mdm2, so that the control of this protein on p53 becomes weaker and p53 can exercise its functions. In case of DNA damage, the concentrations of protein p53 and Mdm2 have an oscillating trend.

Mammalian cell cycle (Zámborszky at al. 2007, [42]). The mammalian cell cycle is traditionally divided into four distinct phases: (i) the synthesis phase, which is the period of DNA replication, (ii) the G2 phase, which is the temporal gap between synthesis and mitosis, (iii) the mitosis phase, when replicated DNA molecules are segregated to daughter cells, and (iv) the G1 phase, which is the temporal gap between mitosis and synthesis. The cell cycle is regulated by several checkpoints, which are moments when the cycle progression is stopped to verify the state of the cell and, if needed, to repair it before damaged DNA is transmitted to progeny cells. The proper alternation among the phases of the cell cycle is coordinated by a family of key proteins

called cyclins. Roughly speaking, each phase is characterized by the high concentration of one of these proteins.

Circadian clock (Leloup et al. 2003, [49]). The circadian clock is a biological clock that regulates the synchronous progression of cells through each stage of the cell cycle. The model we consider mainly describes the negative feedback loop between two protein complexes. This loop originates some sustained oscillations with a period of approximately 24 hours.

The four models we consider consist of ordinary differential equations and are established models in the literature. As a matter of fact, we *re-use already existing models in a systematic way*, which is a challenging issue in systems biology. In fact, many models are still developed, refined, simplified or coupled with respect to other models by hand, with no direct support from the tools to re-use models in a systematic way.

As a first step, we encode the four models in Biocham [24], a programming environment that allows to model bio-chemical systems thanks to a rule based language. As a second step, we draw them into a *coupled model*. In a succinct way, the expected behaviour of the coupled model is the following one (see Figure 2.9): irinotecan injections cause DNA damage; this triggers the action of protein p53, which blocks the cell cycle at a checkpoint; if DNA damage is not too serious, protein p53 repairs it, otherwise the cell is led to die (apoptosis).

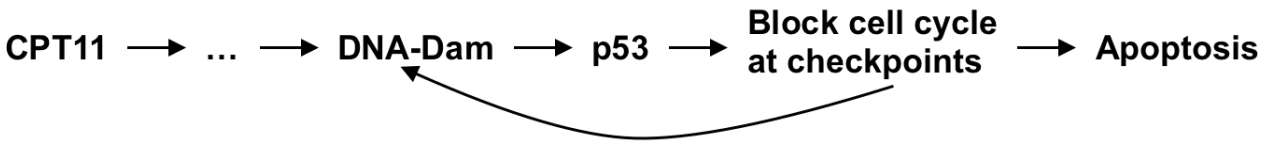


Figure 2.9: Expected behaviour of the coupled model.

To couple the four models, we do not only juxtapose their rules, but we decide how to make them interact. More precisely, we complete the models with some linking rules and find suitable values for the new kinetic parameters so that some specifications are satisfied. To this aim, we review the literature about known links among the four building blocks. For instance, in the literature we find evidence of the fact that, if irinotecan is injected in a cell during the synthesis phase of the cell cycle, then more DNA damage is caused with respect to the other phases of the cell cycle. We provide a characterization of the synthesis phase in terms of the concentration level of a cyclin and we insert in the irinotecan model a dependence from the synthesis phase of the kinetic parameter involved in the production of DNA damage: such a parameter takes a high value during DNA replication and a low value out of synthesis. This links the models of cell cycle and inirotecan. Links between the other models are added in a similar way.

Discrete simulation traces of Biocham models can be easily obtained by means of numerical integration methods (considering the corresponding systems of ordinary differential equations). To apply formal methods, we consider transition systems which are simulation traces over a time window of 100 hours extended with the first derivatives of the involved

variables. To query simulation traces, we use the logic constraint-LTL [22], which extends LTL with arithmetic operators and allows to reason not only on concentration values but also on their derivatives. We also exploit a *parameter learning procedure* that allows to find parameter values that make specifications true. This procedure consists in computing a violation degree for each formula (euclidian distance between the formula and its validity domain), and using such a violation degree as a cost function for a stochastic optimization algorithm.

More precisely, we use formal methods to perform these three different steps.

Specify the linking rules. We look for the kinetic parameters underlying the linking rules such that some given temporal logic formulae are satisfied. While searching for parameters for the new linking rules, we are led to adjust the already known parameters, and thus to modify/improve the single sub-models.

Validate the links. Once we have obtained our coupled model, we check whether some given expected temporal logic formulae are verified by the coupled model.

Find optimal drug control laws. We consider healthy cells and look for irinotecan injection times and maximum amount that keep DNA damage under a given threshold (this is expressed by a temporal logic formula). Then we keep the amount of injected irinotecan constant and maximize toxicity, getting a 12-hours phase shift in injection times. With this phase shift, which could be attained by unynchronized cells, DNA damage increases of 70%.

This contribution lays the groundwork for the automation of model coupling and research in the field of chronotherapeutics. In [3], the authors point out the need for diagnostic and delivery algorithms enabling treatment individualization of cancer patients. They review recent works, including our one, with emphasis on circadian timing system-resetting strategies for improving chronic disease control and patient outcomes.

Remark. For the sake of compactness, I selected a limited number of post-doctoral works. I would like to cite at least another work related to the study of the DNA damage repair system:
Elisabetta De Maria, Joëlle Despeyroux, and Amy P. Felty. *A Logical Framework for Systems Biology.* Proceedings of Formal Methods in Macro-Biology (FMMB 2014), LNCS 8738: 136-155, (2014). *Acceptation rate: 58%.* In this work, we use a modal linear logic to encode a model of the P53/Mdm2 DNA damage repair mechanism and several properties that are important for such a model to satisfy. We formalize the proofs of these properties in the Coq Proof Assistant, with the help of a Lambda Prolog prover for partial automation of the proofs.

2.7.2 Parameter Learning for Spiking Neural Networks Modelled as Timed Automata

My next selected contribution is:

Elisabetta De Maria, Cinzia Di Giusto, and Laetitia Laversa. *Spiking neural networks modelled as timed automata: with parameter learning.* Natural Computing, 1-21, https://doi.org/10.1007/s11047-019-09727-9, (2019).

The paper is reported in Chapter 5. In this work we propose a novel technique to infer parameters of biological neural networks. We deal with *Spiking Neural Networks* (SNNs) [30], which are also referred as third generation networks. They are actually considered closer to the brain functioning than other generation models, and this is mainly due to the fact that they carefully take into account time-related aspects. We show how SNNs can be mapped into timed automata and how this formalization can be exploited to learn the parameter values of a neural network such that the network can display a given behaviour.

Spiking Neural Networks can be seen as directed graphs whose nodes represent neurons and whose edges represent synaptical connections. A weight is associated with each edge: positive (resp. negative) weights stand for excitatory (resp. inhibitory) synapses. Neurons in a network can be distinguished into input neurons, which can only receive as input external inputs, intermediary neurons, and output neurons, whose output is considered the output of the network. Several spiking neural models exist: in this work (and in the next selected contributions) we focus on the *Leaky Integrate and Fire* (LI&F) model [48], which is a computationally efficient approximation of a single-compartment model and is abstracted enough for the application of formal techniques. According to this model, the dynamics of each neuron is governed by its (membrane) potential, which represents the difference of electrical potential across the cell membrane. Each neuron emits a spike whenever its potential value exceeds a given firing threshold. We present a discrete version of the LI&F model: at each time unit, the potential value of a neuron can be computed as a function of its current weighted inputs and of its previous time unit potential value (weakened by a leak factor). After each spike emission, the neuron potential is reset to zero. This model also allows to take into account the refractory period, a lapse of time immediately following the spike emission in which the neuron is quiescent (i.e., the neuron cannot emit spikes).

As a first step, we *map neural networks into timed automata networks*. Timed automata are finite-state machines extended with clock variables that allow to measure time. It is possible to define the synchronous product of a set of timed automata that work and synchronize in parallel. This leads to a network of timed automata, whose component automata can synchronize thanks to some input and output labels. We provide a full implementation of neural networks as timed automata networks via the Uppaal tool [4], which allows to encode and simulate timed automata networks and provides an integrated model checker to automatically verify whether some given temporal logic formulae are satisfied by the encoded systems.

Given a neural network consisting of a set of input, intermediary, and output neurons, we add a set of *input generator* neurons, which are fictitious neurons connected to input neurons that generate input sequences for the network, and a set of *output consumer* neurons, which are fictitious neurons connected to the output of each output neuron that aim at consuming their emitted spikes (see Figure 2.10). To obtain the corresponding timed automata network,

we build an automaton for each neuron of the network and we consider the synchronous product of the obtained automata.

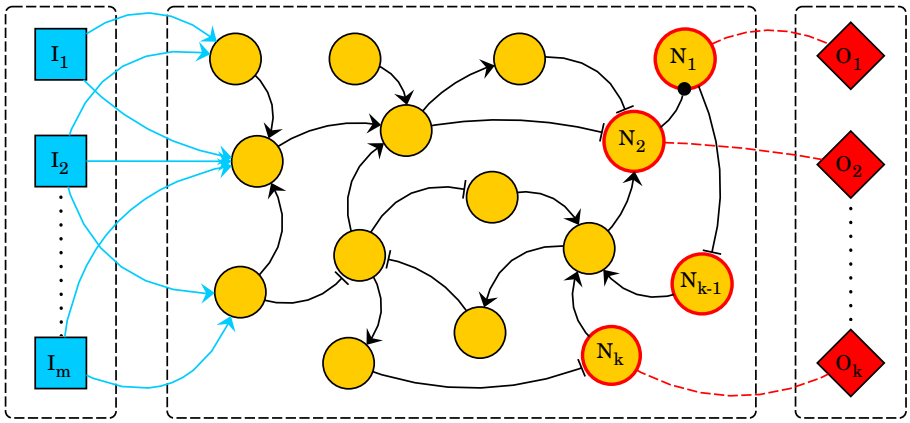


Figure 2.10: Neural network extended with input generator and output consumer neurons.

In [41], the author proposes a classification of the main spiking neuron models according to some behaviours they are able to reproduce or not (a behaviour is a typical response to an input pattern). The LI&F model is supposed to verify three of these behaviours (tonic spiking, excitability, and integrator). To show that our mapping is correct, we prove that our model satisfies these three behaviours and does not satisfy the other ones (behaviours are formalized as temporal logic formulae).

Once we have an encoding of neural networks which is formally validated, we can exploit it to examine the *learning problem*, which consists in finding a parameter assignment for a LI&F network with a fixed topology and a given input such that a desired output behaviour is displayed. The parameters we focus on in this work are the synaptical weights of the network. For the algorithm we propose, we borrow inspiration from a variant of the back-propagation algorithm [70], which aims at training networks to produce a given output sequence for each class of input sequences. We also rest on the spike timing dependent plasticity rule [73], which aims at adjusting the synaptical weights of a network according to the time occurrences of input and output spikes of neurons.

In the algorithm we develop, the learning process is led by some *supervisors*, which are connected to output neurons. Supervisors compare the expected output behaviour with the actual behaviour of the output neuron they are connected to. Thus either the neuron behaves consistently or not. In the second case, the supervisor back-propagates advices to the output neuron depending on two possible scenarios: (i) the neuron remains quiescent, but it was supposed to fire a spike; (ii) the neuron fires a spike, but it was supposed to be quiescent. In the first (resp. second) case, the supervisor addresses a *should have fired* (resp. *should not have fired*) advice. Then each output neuron modifies its ingoing synaptical weights and in turn behaves as a supervisor with respect to its predecessors, back-propagating the proper advice. The algorithm basically lies on a backward depth-first visit of the graph. When the

advices reach input generators, they are ignored. The process ends when all the supervisors do not detect any more errors.

We propose two different implementations for our learning algorithm. The first one, which is *model-checking* oriented, consists in iterating the learning process until a desired temporal logic formula is verified. At each step of the algorithm, we make an external call to a model checker to test whether the network satisfies the formula or not. If the formula is verified, the learning process ends; otherwise the model checker provides a trace as a counter-example; such a trace is exploited to derive the proper corrective action for each neuron, *should have fired* or *should not have fired*. With the second approach, which is *simulation-oriented*, the synaptical weights are modified during the simulation of the network. In this case, supervisors are defined as timed automata. After the simulation starts, supervisors expect a certain behaviour from the output neurons they are connected to. If the behaviour matches, the simulation proceeds; otherwise a proper advice is back-propagated in the network. When the advice reaches input generators, the simulation is restarted from the beginning with the modified weight values. The process ends when all the supervisors detect that output neurons learned to reproduce the proper outcome.

Observe that formal methods play a crucial role in our approach:

- they allow us to *validate the mapping* of neural networks into timed automata networks;
- they are an *integral part of our learning algorithm* (expected behaviours are expressed as temporal logic formulae in the model checking approach, as timed automata in the simulation approach).

In this work, we treat several case studies dealing with non trivial networks featuring excitatory and inhibitory edges and cycles (e.g., a mutual inhibition network, where each neuron inhibits all the other neurons).

Our novel technique to infer the synaptical weights of SNNs adapts machine learning techniques and formal methods to bio-inspired models. This makes our approach original and complementary with respect to the main other projects aiming at understanding the human brain, such as the Human Brain Project [18], which mainly relies on large scale simulations. Future work on this project includes the definition of sophisticated supervisors allowing to compare the output of several neurons and the generalization of the learning procedure to the other key parameters of biological neural networks (firing threshold, leak factor, etc).

2.7.3 Formal Methods to Model and Verify Neuronal Archetypes

In this subsection I focus on my contributions in the field of formal modelling and verification of neuronal archetypes, which are associated with the following three works:

(i) Elisabetta De Maria, Alexandre Muzy, Daniel Gaffé, Annie Ressouche, and Franck Grammont. *Verification of Temporal Properties of Neuronal Archetypes Modeled as Synchronous Reactive Systems.* Proceedings of Hybrid Systems Biology (HSB 2016), LNCS 9957: 97-112, (2016).

(ii) Elisabetta De Maria, Thibaud L'Yvonnet, Daniel Gaffé, Annie Ressouche, and Franck Grammont. *Modelling and Formal Verification of Neuronal Archetypes Coupling.* Proceedings of 8th International Conference on Computational Systems Biology and Bioinformatics (CSBIO 2017), ACM, 3-10, (2017);

(iii) Abdorrahim Bahrami, Elisabetta De Maria and Amy Felty. *Modelling and Verifying Dynamic Properties of Biological Neural Networks in Coq.* Proceedings of 9th International Conference on Computational Systems Biology and Bioinformatics (CSBIO 2018), ACM, Article No. 12, 1-11, (2018).

These works are the fruit of a cooperation with the neurophysiologist Franck Grammont, Laboratoire J.A. Dieudionné, Nice, France. In Chapter 6, I report, extend, and compare these three works (the version of the paper reported in Chapter 6 has been submitted to the journal Frontiers of Computer Science). The biological starting hypothesis is that, in our brain, neurons tend to form some mini-circuits with recurrent structures that we call *archetypes*. Each archetype presents a particular class of behaviours and several archetypes can be coupled to constitute the elementary building blocks of bigger neural circuits. For instance, we know that the different walking ways in animals are controlled by some mini-circuits, called Central Path Generators [58], which are able to generate some oscillations.

The aim of our works is to *formally study the behaviour of a significant set of basic archetypes and their couplings*. Archetypes can be metaphorically seen as the syllables of a given alphabet. The composition of two syllables can give either an existing or a non existent word. At the same way, the composition of two archetypes can give a network displaying a behavior which is biologically meaningful or not. Again, two different syllable compositions can give two words with the same meaning: at the same way, two different archetype compositions can present the same biological behaviour.

The basic archetypes we study are the following ones (see Fig. 2.11): (i) *Simple series:* it is a sequence of neurons where each element of the chain receives as input the output of the preceding one; (ii) *Series with multiple outputs:* it is a series where, at each time unit, we are interested in knowing the outputs of all the neurons; (iii) *Parallel composition:* it is a set of neurons receiving as input the output of a given neuron; (iv) *Negative loop:* it is a loop consisting of two neurons: the first neuron activates the second one while the latter inhibits the former one; (v) *Inhibition of a behavior:* it consists of two neurons, the first one inhibiting the second one; (vi) *Contralateral inhibition:* it consists of two or more neurons, each one inhibiting the other ones.

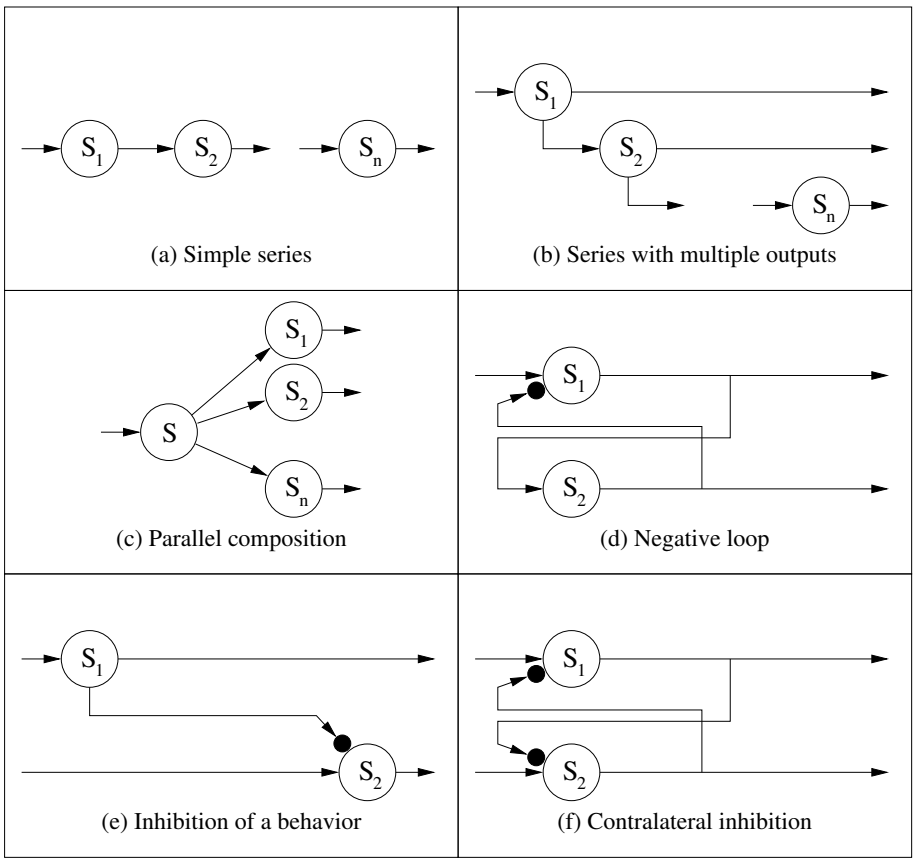


Figure 2.11: The basic neuronal archetypes.

As in the previous subsection, we rely on Spiking Neural Networks where the neuron electrical properties are described via the LI&F model. Neural networks can be seen as *reactive systems*, which are systems that constantly interact with the environment and which may have an indefinite duration. The behaviour of neural networks can indeed be represented as a sequence of reactions to some stimuli. The *synchronous* paradigm is suited to model reactive systems. It is based on the notion of logical time: time is considered as a sequence of discrete instants. At each instant, it is possible to observe some inputs, to make some computations, and to produce some outputs. In the first two cited papers, we use the synchronous language *Lustre* to encode neural networks and to express some expected properties concerning their dynamical evolution. Then model checkers allow us to test whether the formalized properties are satisfied (for some given parameter classes). More precisely, in the first work we deal with neurons and archetypes, and in the second work we make a step further and focus on archetype coupling. We consider two ways to couple two archetypes: concatenation, where the output(s) of an archetype is (are) connected to the input(s) of another archetype, and nesting, where one archetype is nested inside another archetype. Our results show that archetype coupling can either modulate the behaviors displayed by the single archetypes (e.g. extend an oscillation period), or clearly give rise to new behaviors. Furthermore, several different couplings turn out to display the same behavior (whatever

their input sequences are).

In the third work, we rely on the *Coq Proof Assistant* to prove important properties of neurons and archetypes. Coq implements a highly expressive higher-order logic in which we can directly introduce data types modeling neurons and archetypes, and express properties about them. As a matter of fact, one of the main advantages of using Coq is the generality of its proofs. Using such a system, we can prove properties about arbitrary values of parameters, such as any length of time, any input sequence, or any number of neurons. As a drawback, these properties could not be proved automatically, and our expertise was required.

Formal methods of computer science are thus necessary in the above-mentioned works to *formalize and prove properties concerning the time evolution of neurons, archetypes, and their couplings.*

As a short-time goal, we intend to translate our implementations of neurons and archetypes into the VHSIC Hardware Description Language (VHDL) to make it run on a field-programmable gate array (FPGA) [26], that is, an integrated circuit configurable by the customer. This would allow to have a real physical implementation of our models validated through formal techniques.

As a long-term goal, we would like to substantiate the claim that whatever neural circuit, even complex, can be expressed as a composition of archetypes, as far as all words can be expressed starting from a given set of syllables on an alphabet.

2.7.4 A Model Checking Approach to Reduce Spiking Neural Networks

The last contribution I decided to highlight is:

Elisabetta De Maria, Daniel Gaffé, Cédric Girard Riboulleau and Annie Ressouche. *A Model-checking Approach to Reduce Spiking Neural Networks.* Proceedings of 9th International Conference on Bioinformatics Models, Methods and Algorithms (BIOINFORMATICS 2018), 89-96, (2018).

The corresponding paper appears in Chapter 7. In this work we introduce a *new algorithm for reducing the number of neurons and synaptical connections of a given neural network.* The proposed reduction preserves the desired dynamical behavior of the network, which is formalized by means of PCTL temporal logic formulae and verified thanks to the PRISM model checker [47].

We choose to rely on the discrete Leaky Integrate and Fire (LI&F) model already introduced in the previous subsections, but this time the LI&F model is augmented with *probabilities.* More precisely, the probability for a neuron to emit a spike is driven by the difference between its membrane potential and its firing threshold. We discretize the difference between the membrane potential and the firing threshold into a certain number of positive and negative intervals and we associate a probability to each interval. For positive (resp. negative) values of this difference, the higher its absolute value is, the higher (resp.

lower) the probability to emit a spike is. Probability values are chosen in order to conform to a sigmoidal function.

Thanks to the modeling language at PRISM user's disposition, LI&F probabilistic networks are encoded as Discrete-Time Markov Chains (DTMCs), which are transition systems augmented with probabilities. Their set of states represents the possible configurations of the system being modeled, and transition between states model the evolution of the system, which occurs in discrete-time steps. Probabilities are associated to transitions.

The reduction algorithm we propose supposes the neural network at issue to be implemented as a DTMC in PRISM, and makes several calls to the PRISM model checker to retrieve the probability relative to the satisfaction of some temporal logic formulae. The algorithm takes as input a LI&F network and a PCTL property concerning the dynamical behaviour of the output neurons of the network. Only intermediary neurons are affected by the reduction process, that is, input and output neurons cannot be removed. As a first step, intermediary neurons are visited following a depth first search in order to remove *wall* neurons, that is, neurons that are never able to emit, even if they receive a persistent sequence of spikes as input. When the algorithm detects a wall neuron, it removes not only the neuron but also its descendants whose only incoming synaptical connection comes from this neuron, and its ancestors whose only outgoing edge enters the neuron. As a second step, the algorithm performs another DFS traversal of the remaining intermediary neurons to identify and remove neurons whose removal has a law influence (i) on the probability for the PCTL property to be satisfied and (ii) on the spike rate. As in the previous step, descendants and ancestors of these neurons are consistently removed.

The use of formal methods is thus crucial to:

- *formalize the expected behaviour of the network* (as a PCTL property);
- *automatically retrieve the probability for the behaviour to be displayed* when removing some neurons (thanks to several calls to the PRISM model checker).

Our experiments show that the application of this algorithm can drastically reduce the number of states and transitions of the transition system corresponding to the DTMC at issue. For instance, we show that the removal of one neuron from a network of four neurons reduces (in average) the number of states and transitions of a factor 20.

The actual version of the algorithm finds a reduction which conforms to the expected behaviour of the network. It is not necessarily the optimal solution, that is, it does not necessarily minimize the difference of behaviour between the complete and the reduced network, and we plan to address this issue in our future work.

Observe that, besides its utility in lightening models, our algorithm for neural network reduction has a forthright application in the medical domain. In fact, it can help in detecting weakly active (or inactive) zones of the human brain.

Chapter 3

Formal Methods for Systems Biology: Position on Current and Future Directions of Research

3.1 What is the Best Formal Method for Systems Biology?

During my post-doctoral researches I had the chance to apply formal methods to make advanced analysis on several biological systems. In particular, I could experiment the use of the two most common approaches for formal verification: the *model checking* approach and the *theorem proving* approach. The study of biological neural networks gave me the possibility to directly compare the benefits of these two different techniques for the formalization and proof of dynamical properties of neuronal archetypes.

In absolute terms, I could not say one of the two approaches is strongly preferable with respect to the other for the formal study of the dynamics of biological systems. As already explained, the main advantage of the model checking methodology is that it is completely automatic: once a given property has been correctly encoded, the user just needs to press a button to know whether the property is verified or not. So the presence of an expert is not needed to obtain a proof. Another strength of model checking is that, in case a property does not hold in a given model, a counter-example is automatically provided. Such an execution trace can give hints in understanding what should be modified in the system so that the property is satisfied. Furthermore, even if the transition system corresponding to each model is exponential with respect to the number of variables, several model checkers exploit some advanced features to improve scalability, and their answers are often immediate for our models and properties.

On the other hand, model checkers often cannot prove properties at the desired level of generality. As a matter of fact, the use of a proof assistant guarantees that the properties we prove are true in the general case, such as true for any input values, any length of input, and any amount of time. Another advantage of the theorem proving approach is that, since

we have access to proofs, a successful proof of a given property can be exploited to prove similar properties. The drawbacks are that proofs can be long and an expert is needed to make them.

My feeling is that, to perform formal advanced analysis of biological systems, the model checking and theorem proving approaches should be used together in a pragmatic way. When trying to prove a given property, the idea is to first test its validity for some crucial given parameter intervals using model checking, and eventually refine the model thanks to the provided counter-examples so that the property holds in the defined context. Once some key tests have passed, the theorem proving technique can be exploited to prove the property in a more general context. The complementarity of these two powerful techniques could allow significant advances in the study of the dynamics of biological systems.

3.2 Hot Issues: Model Composition and Model Reduction

As the reader may have noticed, *model composition* and *model reduction* are two key issues playing an important role in all my post-doctoral researches.

3.2.1 Model Composition in my Works

As far as model composition (or coupling) is concerned, it is at the heart of [51], where we propose a coupled model of the cell cycle, the circadian clock, the DNA repair system, and the irinotecan metabolism and exposure control. One of the main aims of this work is to show how the *validation of a coupled model and the optimisation of its parameters* (with respect to biological properties of the coupled system) can be done *automatically* using model checking and parameter learning techniques. To couple our models, we complete them with some linking rules and find suitable values for the kinetic parameters such that some specifications are satisfied. Notice that it is often not possible to find parameter values for the new kinetic rules without changing some parameters in the original models. Thus *coupling models can help in better understanding and improving the original models*. Furthermore, *coupling models can help in detecting some lacks in the original models*. For instance, a missing metabolic pathway in a model can be detected by coupling it with another model.

Model composition is also at the basis of [54], where we compose (couple) several neuronal archetypes to study the behavior of the resulting network. In this context, crucial questions underlying our study are:

- Are the properties of the resulting network simply the conjunction of the individual constituent archetype properties or something more? In other words, does the resulting network satisfy only properties that were already satisfied by the constituent archetypes or are there new properties that are also satisfied by this composition?

- Can we understand the computational properties of large groups of neurons simply as

the coupling of the properties of individual archetypes, as it is for syllables and words, or is there something more again?

Model composition is present in [53] too, even if in a less explicit way than in the previous cited works. In this work, we model biological neurons as timed automata and we compose them to obtain timed automata networks whose neurons work and synchronize in parallel (formally, this can be seen as the synchronous product of a set of timed automata).

3.2.2 Model Reduction in my Works

As far as *model reduction* is concerned, it *is crucial in systems biology, especially in the scope of the attainment of models which are suitable for formal verification.* Model reduction motivates our work in [52], where we propose an algorithm to reduce the number of nodes and edges of a given biological neural network while preserving the desired dynamical behavior of the network. Roughly speaking, a node (and its incoming and outcoming edges) is removed if its deletion has a low impact on the probability for a given temporal logic formula to hold. Such an algorithm could be easily adapted to the reduction of other kinds of biological networks.

Notice that we also cope with model reduction in [54], where we deal with biological neuronal archetypes. In this work we write a Lustre observer to verify whether two given networks always display the same behaviour, whatever their input sequences are. For instance, we exploit such an observer to verify whether two simple series of different length have the same output sequences, whatever their input sequences are. A short series having the same behaviour than a longer series can replace the former one in a biological network, thus contributing to reduce the size of the network.

We also deal with model reduction in [53], even if less explicitly than in the other cited works. In this work, we propose an algorithm to infer the synaptical weights of a given network such that the network can display a given behaviour. This algorithm may lead us to set some synaptical weights to zero, which corresponds to removing some connections, and thus lightening the initial network.

3.2.3 Model Composition and Model Reduction in Systems Biology

Model composition and model reduction are central issues in systems biology, and techniques to automatize these processes are more and more needed. Of course our works are not the only ones attacking these topics. Concerning model composition, in [17] the authors propose a web-based solution to facilitate the process of merging biological models encoded in SBML (Systems Biology Markup Language), and in [32] the authors apply system engineering methods to compose continuous models of biological systems. Concerning model reduction, emblematic examples can be found in [59], where the authors propose a methodology to reduce regulatory networks preserving some dynamical properties of the original models, such as stable states, in [28], where the authors study model reductions as graph matching problems, or in [62], whose author considers finite-state machines and proposes a

technique to remove some transitions while preserving all the (minimal) traces satisfying a given reachability property.

My feeling is that model composition and model reduction will be at the heart of deep researches in the next years, and I intend to continue to contribute to the study of these hot topics with the help of formal methods. As for model composition, *one of the most hard tasks will be to formulate the properties of the composed models starting from the properties of the constituent models,* which is often not straightforward. As for model reduction, *the main difficulty may consist in selecting the most suited reduction when several alternatives exist.*

3.3 Current Work and Perspectives

In the previous sections I described my main contributions and ideas in the field of formal methods for systems biology. I would like to underline that the works I presented and the opinions I exposed derive from the strong interactions I had (and continue to have) with my master and Ph.D. students and from chairing three conferences in the field of systems biology and computational biology more in general (BIOINFORMATICS 2019, CSBio 2019, BIOINFORMATICS 2020). These scientific duties gave me the possibility to read in detail a big number of papers, to listen to the corresponding presentations, and to compare them to make some critical decisions (choice of the best papers, selection of papers for publication in a journal, review of extended versions, etc). This allowed me to get a broad-spectrum view on systems biology, which goes beyond the research topics I specialized on, and which I believe will be useful for the follow-up of my researches.

As far as my current and future researches are concerned, my firm intention is to continue to exploit formal methods to perform a deep analysis of biological neural networks. As already mentioned, the use of formal techniques makes my researches quite original and complementary with respect to the main international projects aiming at understanding the brain functioning, which are mainly based on large systems of differential equations [18].

My main long term project consists in focusing on neuronal archetypes and try to substantiate the theory that whatever biological neural network can be expressed as a composition of some canonical archetypes. I am currently working on this topic with the associate professor Franck Grammont from Laboratoire J.A. Dieudionné, Nice, France, the Ph.D. student Abdorrahim Bahrami, and the full professor Amy Felty from University of Ottawa, Canada. Abdorrahim Bahrami is supposed to defend his Ph.D. thesis by the end of Autumn 2020, and to keep focusing on the formal study of neuronal archetypes thanks to a post-doctoral contract under my supervision. Concerning the study of biological neural networks, I also intend to keep working on the development of algorithms for both reducing the size of networks and automatically finding their crucial parameters. For this last point, a combination of formal methods and machine learning techniques, as described in [53], seems to be highly suited.

Another research axis in which I started to be involved recently deals with bio-medicine and results from the co-supervision of the student Thibaud L'Yvonnet, who started his

Figure 3.1: Display of the Match Items game.

Ph.D. program in December 2019. In this Ph.D. thesis, we deal with "serious games" for health, which are used by medical doctors to evaluate the performances of patients affected by neuro-degenerative pathologies such as the Alzheimer disease [79]. Behavior, emotions, and performance displayed by patients during these games can indeed give indications on their disease. An example of serious game to analyze the behavior of Alzheimer patients is the *Match Items game* (see the screenshot in Figure 3.1). In this game, patients interact with a touch-pad. They are asked to match a random picture displayed in the center of the touch-pad with the corresponding element in a list of pictures.

In the first part of the Ph.D. thesis, we proposed a formal approach to model serious games, taking into account possible variations in human behavior. Starting from an activity description enriched with event occurrence probabilities, we translate it into a corresponding formal model based on discrete-time Markov chains [55]. We use the PRISM framework and its model checking facilities to express and test interesting temporal logic properties (PCTL) concerning the dynamic evolution of activities. In particular, we retrieve probabilities for the patients to follow some given crucial classes of paths. This phase is necessary to validate our approach and to explore the kind of properties that model checking can achieve, before performing clinical tests on real patients.

We intend now to validate our formal approach thanks to four serious games selected with the help of the team of the medical doctor Philippe Robert from Institut Claude Pompidou, Nice, France. These serious games have been represented with PRISM models, and will be used in clinical experimentation. The configuration for different reference profiles (such as Mild, Moderate, or Severe Alzheimer) will be set up with the participation of clinicians. Then, several groups of patients will play these games and their results will be recorded. We will compare these results with the ones of our models to calibrate the models and to perform new targeted clinical tests.

Each game has been chosen because it targets a specific cerebral function (e.g., inhibitory control or episodic memory), and in the last part of the thesis we will compare the results obtained in the previous phase with some biological neural networks describing the cerebral

functions targeted by the games. This last part of the thesis, more prospective, has two purposes: (i) study how to model some given neural networks to keep some deviating behaviors into account (which parameters to change and how), and (ii) predict the behaviours associated to some specific neuronal configurations. This part of the work, where we try to cross behavioral models and brain models, is more ambitious and well fits with my strong will of better understanding the behaviour of the human brain. There will certainly be some intersections between this work and the study of neuronal archetypes I am going to conduct.

Chapter 4

Design, Optimization and Predictions of a Coupled Model of the Cell Cycle, Circadian Clock, DNA Repair System, Irinotecan Metabolism and Exposure Control under Temporal Logic Constraints

Elisabetta De Maria, François Fages, Aurélien Rizk, and Sylvain Soliman
Journal of Theoretical Computer Science, 2011

Design, Optimization and Predictions of a Coupled Model of the Cell Cycle, Circadian Clock, DNA Repair System, Irinotecan Metabolism and Exposure Control under Temporal Logic Constraints[1]

Elisabetta De Maria, François Fages, Aurélien Rizk, Sylvain Soliman

EPI Contraintes, INRIA Paris-Rocquencourt, France

Abstract

In systems biology, the number of available models of cellular processes increases rapidly, but re-using models in different contexts or for different questions remains a challenging issue. In this paper, we study the coupling of different models playing a role in the mammalian cell cycle and in cancer therapies. We show how the formalization of experimental observations in temporal logic with numerical constraints can be used to compute the unknown coupling kinetics parameter values agreeing with experimental data. This constraint-based approach to computing with partial information is illustrated through the design of a complex model of the mammalian cell cycle, the circadian clock, the p53/Mdm2 DNA-damage repair system, the metabolism of irinotecan and the control of cell exposure to it. We discuss the use of this model for cancer chronotherapies and evaluate its predictive power with respect to circadian core gene knock-outs.

Keywords: model coupling, temporal logic, model checking, constraint solving, parameter learning, cell cycle, DNA damage, irinotecan

1. Introduction

In systems biology, the number of available models of cellular processes increases rapidly. To date, most of the effort has been devoted to building models and making them freely available, through the design of standard exchange formats, such as for instance the Systems Markup Language SBML [29], the making of model repositories, such as for instance Biomodels[2], the

[1] This article is an extended version of [17].

[2] `http://biomodels.net/`

making of biological ontologies to establish the links between molecular synonyms, species, units, etc., and the development of modeling tools, such as Cell Designer, Biocham [8], BioNetGen [6], Pathway Logic [19], Bio-ambients [40], etc. Despite these efforts however, re-using models in different contexts or for different questions remains a challenging issue. In practice, most of the models are developed, refined, simplified or coupled with respect to other models by hand with no direct support from the tools to re-use models in a systematic way using a specification of the global behavior of the system.

Coupling biological models is necessary to study how the building blocks interact together and make predictions on the global system's behavior. Model coupling is also a method to better understand and improve the composite models. The knowledge acquired from the global view provided by a coupled model can indeed lead to modify the single model components in order to satisfy some observed property of the global system. In particular, coupling models can help identifying lacks in the model components, like a missing node in a pathway for instance.

In this paper, we show how the formalization of experimental observations in temporal logic with numerical constraints can be used to automatically find parameter values for the coupling kinetics agreeing with experimental data. We illustrate this constraint-based approach to computing with partial information, through the coupling of existing biochemical models of the mammalian cell cycle, the circadian clock, the p53/Mdm2 DNA-damage repair system, and irinotecan metabolism. Finally, we discuss the predictive power of the obtained coupled model with respect to circadian core gene knock-outs.

Mammalian cell cycle

Irinotecan is an anti-carcinogenic inhibitor of topoisomerase-1 which started to be used in clinical treatments approximately twenty years ago [34]. It shows significant efficacy against a variety of solid tumors, including lung, colorectal, and cervical cancers. Scientists are currently trying to optimize the irinotecan therapy in order to understand how to limit its toxicity on healthy cells and to increase its efficacy [2]. In this context, it is crucial to comprehend how the administration of this medicament influences cellular proliferation. For this purpose, the observed effects of the circadian rhythm on the toxicity and efficacy of anti-tumor drugs should be taken into account. In fact, the effectiveness of anti-cancer drugs on a healthy as well as tumorous cells is dependent on the phase of the cell cycle in which those cell lie [2]. Under the hypothesis that the cell cycle in healthy tissues is mainly entrained by the circadian clock, it is possible to reduce the toxicity on healthy cells by injecting antitumor drugs in precise periods of the circadian clock.

On the other hand, tumorous cells are either phase-shifted (slow-growing tumors) or not entrained any more (rapidly growing or advanced stage tumors). A rhythmic drug exposure can thus limit toxicity on healthy cells while maintaining efficacy on tumour cells.

In this paper, we develop a complex model of the mammalian cell cycle, circadian clock, p53/Mdm2 DNA repair system and ironotecan metabolism to investigate the influences of irinotecan on cell proliferation. There are in the literature many models of the mammalian cell cycle [35, 25] and of the circadian biochemical clock [32, 24], a few ones of the cell's DNA-damage repair network [13, 12], and recently some preliminary models of irinotecan intracellular pharmacodynamics [18, 4]. However these modules need to be composed in a coherent way to make meaningful predictions.

Modeling under temporal logic constraints

Our approach to modeling in systems biology consists in formalizing the relevant properties of the behavior of the global system in temporal logic, and in using model-checking, constraint solving and continuous optimization algorithms to compute unknown parameters and validate the model with respect to its temporal specification. This temporal logic based approach is at the heart of our modeling platform, the Biochemical Abstract Machine Biocham [8, 23].

Model-checking is the process of algorithmically verifying whether a given state transition structure is a model for a given temporal logic formula [15]. In the literature, there are now various applications of model-checking techniques to biology. In [10, 19], temporal logic was first introduced as a query language for biochemical networks and for validating boolean models of biological processes. Some experimental results were obtained on a large scale with Kohn's map [31] of the mammalian cell cycle control [11] (800 reaction rules, 500 variables) using the symbolic model-checker NuSMV, and on a small ordinary differential equation (ODE) model using the constraint-based model checker DMC. This approach to verifying biological processes has pushed the development of model-checking techniques for quantitative properties, and continuous, stochastic or hybrid models.

For (non-linear) ODE models, numerical integration techniques provide numerical traces on which formulae of Linear Time Logic with numerical constraints over $\mathbb{R}$, named LTL($\mathbb{R}$) , can also be evaluated by model-checking [7]. Simpathica [3] and Biocham are two computational tools integrating such model-checkers for quantitative models. This approach has been further developed in Biocham by generalizing model-checking to a *temporal logic constraint solving* algorithm [22], allowing for efficient kinetic parameter optimization [41] and robustness analysis [42] w.r.t. quantitative temporal

properties formalized in LTL($\mathbb{R}$) [21].

Related work concerns stochastic models and parameter uncertainty studies. In [28], Heath et al. apply the probabilistic model-checker PRISM to the study of a complex biological system, namely, the Fibroblast Growth Factor (FGF) signalling pathway. In [14], Clarke et al. apply statistical model-checking on a stochastic model of a T-cell receptor. In [5] Batt et al. develop a modeling framework based on differential equations to analyze genetic regulatory networks with parameter uncertainty. The values of uncertain parameters are given in terms of intervals and dynamical properties of the networks are expressed in temporal logic. Model-checking techniques are then exploited to prove that, for every possible parameter value, the modeled systems satisfy the expected properties and to find valid subsets of a given set of parameter values (such an approach is exploited in RoVerGeNe, a tool for robust verification of gene networks). In [37], Piazza et al. propose semi-algebraic hybrid systems as a natural framework for modeling biochemical networks, taking advantage of the decidability of the model-checking problem for Timed Computation Tree Logic.

In this paper, we focus on the use of LTL($\mathbb{R}$) temporal constraints for integrating biochemical models. In order to compose the different modules, we assume a finite set of hypotheses concerning the structure of the links. The unknown kinetic parameter values are then computed by solving the temporal logic constraints using an evolutionary continuous optimization algorithm [41] in order to make the model components interact in a proper way. For this, the biological properties of the global system are formalized as LTL($\mathbb{R}$) constraints, and solved so that the expected properties are automatically satisfied by the coupled model.

Organization of the paper

The paper is organized as follows. In Section 2 we introduce the temporal logic with numerical constraints LTL($\mathbb{R}$) used to specify relevant properties of both the composite models and the coupled model. Section 3 describes the elementary cell processes considered, their models taken from the litterature and their specification in LTL($\mathbb{R}$) . Section 4 presents the coupling of these elementary models and the specification of the global properties of the system in LTL($\mathbb{R}$) . Section 5 gives some performance figures for the evaluation of the parameter optimization method and the inference of parameter values . Then Section 6 shows how the coupled model and the parameter search method can be used to derive an optimal control model for maximizing the volume of irinotecan under non-toxicity constraint in synchronized (healthy) cells. Finally, in Section 7 we illustrate the predictive power of our coupled model by investigating the effects of clock genes knock outs on the cell cycle

in silico and comparing the results with the literature. All the models and the temporal logic formulae used are available in (SBML compatible) Biocham format at [3].

2. Preliminaries on rule-based modeling and LTL($\mathbb{R}$) temporal logic specifications

The Systems Biology Markup Language (SBML) is a widely used rule-based formalism to describe systems of biochemical reactions. SBML is a useful format for exchanging models between modelers, and has been adopted for large repositories of models, such as for instance `biomodels.net`.

The rule-based language of Biocham for describing reaction models is compatible with SBML. Biocham adds a specification language based on temporal logic for formalizing the global properties of the system observed in biological experiments, under various conditions or gene mutations. Having formal languages not only for describing biochemical reaction models, but also for specifying their behavior, opens a whole avenue of research for designing automated reasoning tools to help the modeler [20].

The biological properties of quantitative models can be formalized in Biocham by formulae of the Linear Time Logic with numerical constraints over the reals LTL($\mathbb{R}$) [7, 21, 41]. LTL($\mathbb{R}$) formulae are formed over first-order atomic formulae with equality, inequality and arithmetic operators ranging over real values of concentrations and of their derivatives, using the logical connectives and the usual *temporal operators* of LTL($\mathbb{R}$) : in particular operator **G** for "always in the future", **F** for "sometimes in the future", the next time operator **X**, and the binary operator until **U**.

For instance, $\mathbf{F}([A] > 10)$ expresses that the concentration of A eventually gets above the threshold value 10 and $\mathbf{G}([A] + [B] < [C])$ states that the concentration of C is always greater than the sum of the concentrations of A and B. Oscillation properties, abbreviated as $oscil(M, K)$, are defined as

The abbreviated formula $oscil(M, K, V)$ adds the constraint that the maximum concentration of M must be above the threshold V in at least K oscillations while $period(M, P)$ states that M oscillates at least 3 times and has a period P for the last three oscillations. It is worth noting that this expression of oscillations in temporal logic does not impose us to fix the phase and period of oscillations as in curve fitting.

LTL($\mathbb{R}$) formulae are interpreted in linear state transition structures which represent either an experimental data time series or a simulation trace, both

[3] `http://contraintes.inria.fr/supplementary_material/TCS-CMSB09/`.

completed with loops on terminal states. Given the ODE corresponding to a reaction model, under the hypothesis that the initial state is completely defined, a discrete simulation trace can be obtained by means of a numerical integration method (namely Rosenbrock method for stiff systems). Since constraints refer not only to concentrations, but also to their derivatives, traces of the form

$$(< t_0, x_0, dx_0/dt >, < t_1, x_1, dx_1/dt >, \dots)$$

are considered, where at each time point t_i, the trace associates the concentration values x_i to the variables, and the values of their first derivatives dx_i/dt. It is worth noting that in adaptive step size integration methods of ODE systems, the step size t_{i+1} - t_i is not constant and is determined through an estimation of the error made by the discretization. The notion of *next state* refers to the state of the following time point in a discretized trace, and thus does not necessarily imply a real time neighborhood. The rationale is that the numerical trace contains enough relevant points, and in particular those where the derivatives change abruptly, to correctly evaluate temporal logic formulae.

Beyond verifying whether an LTL($\mathbb{R}$) formula is satisfied in a numerical trace (model-checking), an original algorithm for solving LTL($\mathbb{R}$) constraints [22, 21] has been introduced to compute a continuous satisfaction degree in $[0, 1]$ for LTL($\mathbb{R}$) formulae [41], opening up the field of model-checking to optimization. This is implemented in Biocham using an evolutionnary continuous optimization algorithm [] for optimizing parameter values with respect to LTL($\mathbb{R}$) properties.

3. Elementary Cell Process Models and Temporal Specifications

In this section we introduce the biological processes we deal with, giving temporal logic formulae to specify the behaviour of each of them. Each property is expressed first in natural language, then formalized in LTL($\mathbb{R}$) .

3.1. Mammalian Cell Cycle Control

Cells reproduce by duplicating their contents and then dividing in two. To produce a pair of genetically identical daughter cells, the DNA has to be faithfully replicated, and the replicated chromosomes have to be segregated into two separate cells. The duration of the cell cycle varies greatly from one cell type to another; in many mammalian cells it lasts about 24 hours. The cycle is traditionally divided into the following four distinct phases [1]: the **G1-phase**, that is the temporal gap between the completion of mitosis and

the beginning of DNA synthesis, the **S-phase (synthesis)**, that is the period of DNA replication, the **G2-phase**, that is the temporal gap between the end of DNA synthesis and the beginning of mitosis, and the **M-phase (mitosis)**, when replicated DNA molecules are finally separated in two daughter cells.

The cell cycle is regulated by different checkpoints, that are moments when the cell progression is stopped to verify the state of the cell and, if needed, to repair it before damaged DNA is transmitted to progeny cells. DNA damaging agents trigger checkpoints that produce arrest in G1 and G2 stages of the cell cycle. Cells can also arrest in S, which amounts to a prolonged S phase with slowed DNA synthesis. Arrest in G1 allows repair before DNA replication, whereas arrest in G2 allows repair before chromosome separation in mitosis.

The proper alternation between synthesis and mitosis is coordinated by a complicated network that regulates the activity of a family of key proteins. These proteins are composed of two subunits: a regulatory subunit, a *cyclin*, and a catalytic subunit, the cyclin-dependent kinase, *cdk* for short. A cdk has to associate with a cyclin partner to form a dimer and has to be appropriately phosphorylated in order to be active. The progression through cell cycle is orchestrated by the rise and fall of the Cdk/cyclin dimers which are characteristic of each phase.

In this work we refer to the model of mammalian cell division proposed by Novák and Tyson in [35] and extended by Zámborszky et al. in [45] to include the regulatory activity of Wee1, a kinase that delays or prevents mitosis by phosphorylation of the Cdk1/CyclinB complex. The extended model comprises 22 differential equations and 4 steady-state relations.

This is the specification for a 100-hours simulation of the model:
$\mathbf{F}_{cell}$**:** CycA is greater than 2 in at least 4 oscillations and CycB is greater than 3.5 in at least 4 oscillations and CycD is greater than 0.4 in at least 4 oscillations and CycE is greater than 1 in at least 4 oscillations.
LTL($\mathbb{R}$) : $oscil([CycA], 4, 2) \wedge oscil([CycB], 4, 3.5) \wedge oscil([CycD], 4, 0.4) \wedge oscil([CycE], 4, 1)$.

3.2. Mammalian Circadian Clock

In many living organisms, the activity of some genes and proteins spontaneously display sustained oscillations with a period close to 24 hours. A biochemical clock present in each cell is responsible for maintaining these oscillations at this period. In mammalian cells, two major proteins are transcribed by clock genes in a circadian manner, CLOCK and BMAL1, which in turn bind to form a heterodimer responsible for the transcription of PER (Period) and CRY (Cryptochrome). The two newly-formed proteins then bind

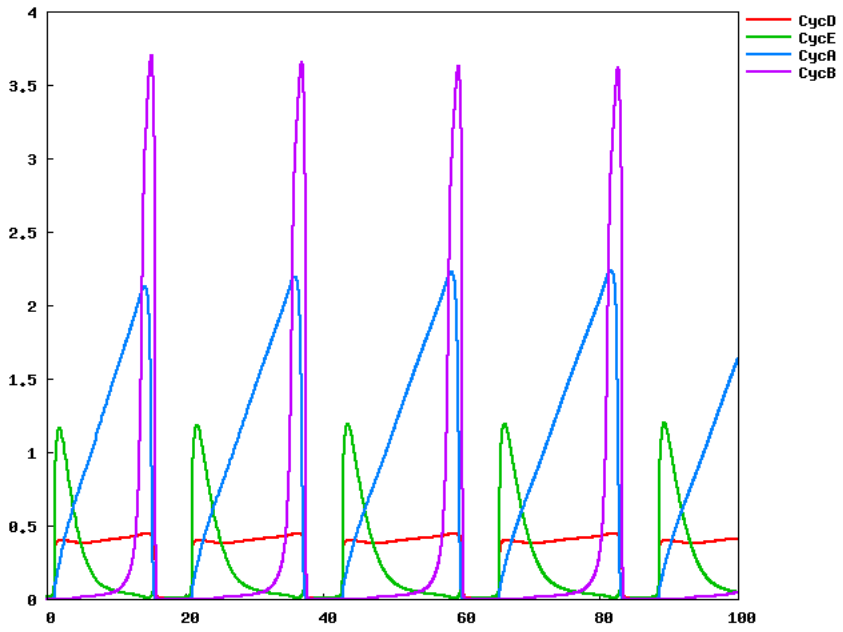


Figure 1: Simulation plot of the cyclin concentrations during the mammalian cell cycle.

as soon as the activity of the complex reaches a threshold. PER/CRY associates with the complex CLOCK/BMAL1 to inhibit its activity and therefore the transcription of the two proteins PER and CRY. This negative feedback loop gives rise to sustained oscillations.

The adaptation of biological organisms to their periodically varying environment is mediated through the entrainment of circadian rhythms by light-dark (LD) cycles. Light can entrain circadian rhythms by inducing the expression of the PER gene.

The model of the circadian clock considered in this work is the one proposed by Leloup and Goldbeter in [32], that consists of 19 differential equations incorporating the regulatory effects exerted on gene expression by the PER, CRY, BMAL1, CLOCK, and REV-ERBα proteins, as well as post-translational regulation on these proteins by reversible phosphorylation, and light-induced PER expression.

The cyclic behaviour of the main compounds of the system is specified by the following formula:
$\mathbf{F}_{clock}$: mPER, mCRY, mBmal1, and mREVERB oscillate with a period equal to 24 (in the last three oscillations).
LTL($\mathbb{R}$) : $period(mPER, 24) \wedge period(mCRY, 24) \wedge period(mBmal1, 24) \wedge$

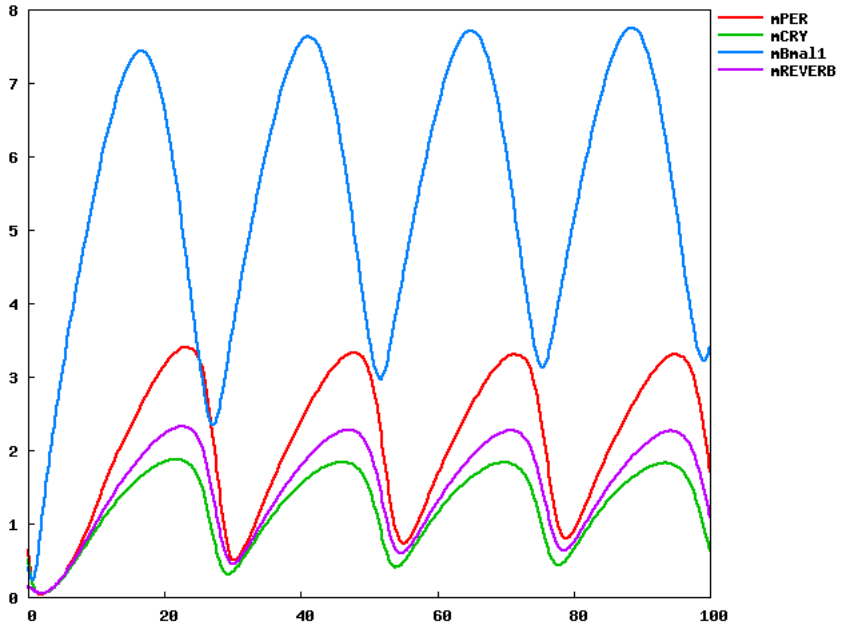


Figure 2: Simulation plot of the mammalian circadian clock genes expression.

$period(mREVERB, 24)$.

Some recent researches showed the existence of biochemical links between the circadian and the cell cycle. In particular, Matsuo et al. [33] proved that a cell cycle regulator, Wee1, is directly regulated by clock components.

3.3. P53/Mdm2 DNA-damage Repair System

The third model is devoted to the description of protein p53, a tumor suppressor protein which is activated in reply to DNA damage. P53 has the capability to arrest the cell cycle in the different phases and to lead to apoptosis, i.e. cell death. P53 can be activated in many ways, in particular in response to DNA damage.

In normal conditions, the concentration of p53 in the nucleus of a cell is feeble: its level is controlled by another protein, Mdm2. These two proteins present a loop of negative regulation. In fact, p53 activates the transcription of Mdm2 while the latter accelerates the degradation of the former. DNA damage increases the degradation rate of Mdm2 so that the control of this protein on p53 becomes weaker and p53 can exercise its functions. This protein is responsible for the activation of many mechanisms: in an indirect way, it stops the DNA synthesis process, it activates the production of proteins

charged with DNA reparation, and can lead to apoptosis.

When DNA is damaged, Mdm2 looses its influence on p53 and one can observe oscillations of p53 and Mdm2 concentrations. The response to a stronger damage is a higher number of oscillations. Oscillations have a very regular period. In literature, several models have been proposed to model the oscillatory behaviour of proteins p53 and Mdm2, most notably the ones proposed by Chickermane et al. [12], by Ciliberto et al. [13], and by Geva-Zatorsky et al. [26]. In this work we build upon the one described in [13], that consists of 6 differential equations.

The following three properties concern the behaviour of proteins p53 and Mdm2.

$\mathbf{F}_{1\_p53}$**:** In case there is no DNA damage, p53 and Mdm2 are constant functions.

LTL($\mathbb{R}$**)** $:\mathbf{G}([DNAdam] = 0) \rightarrow \mathbf{G}(d([p53])/dt = 0 \wedge d([Mdm2 :: n])/dt = 0)$.

$\mathbf{F}_{2\_p53}$**:** Sustained DNA damage causes at least one oscillation of proteins p53 and Mdm2.

LTL($\mathbb{R}$**)** $:\mathbf{G}(([DNAdam] > 0.2) \rightarrow \mathbf{F}(oscil([p53], 1) \wedge \mathbf{F}(oscil([Mdm2], 1))))$.

$\mathbf{F}_{3\_p53}$**:** p53 oscillations are alternated by Mdm2 ones.

LTL($\mathbb{R}$**)** $: \mathbf{G}(oscil([p53], 1) \rightarrow \mathbf{X}((\neg oscil([p53], 1))\mathbf{U}(oscil([Mdm2 :: n], 1))))$.

3.4. Irinotecan Metabolism

Camptothecins are substances that can be extracted from the Chinese tree "Camptotheca acuminata Decne" and are mainly used for the treatment of digestive cancers. Their anticancerogenic properties have been discovered at the end of the Fifties but the first clinical tests have been interrupted owing to heavy effects due to the toxicity of the substances. In the Eighties researchers discovered that camptothecins are inhibitors of topoisomerase-1 (Top1 for short), an essential enzyme for DNA synthesis. Afterwards, they started to focus on some semi-synthetic derivative of water-soluble camptothecins, such as irinotecan and topotecan. Irinotecan is pro-medicine and must be transformed in its active metabolite, SN38, to be effectively cytotoxic. In fact the anticancerogenic activity of irinotecan (CPT11) is approximately 100 times less effective than the one of SN38. The activation is due to carboxylesterase, an enzyme mainly located in the liver, in the intestine, and in the tumoral tissues. SN38 is then detoxified through glucorono-conjugation: this realizes uridine diphosphate glucoronosyl transferase 1A1.

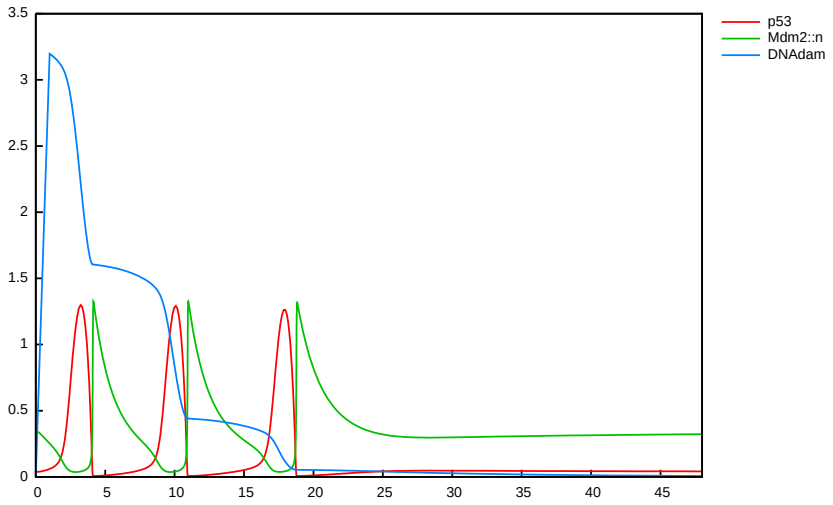


Figure 3: Simulation plot of the P53/Mdm2 DNA-damage repair system.

Mechanisms through which irinotecan damages the cell are very complex and have not been completely explained yet. It is sure that DNA lesions appear after the inhibition of Top1 by SN38. Top1 is a protein which is present in all living organisms and which checks DNA replication and transcription. It intervenes to modify the DNA winding degree, acting on one strand. More precisely, Top1 links itself to the extremity 3' of DNA forming a transitory cleavage complex and cuts a DNA strand, that in such a way is able to unroll. Then such a complex dissociates and a new ligature comes up. In normal conditions, the connection process is favored with respect to the cleavage one. The target of irinotecan, and above all of its active metabolite SN38, is the complex Top1-DNA. SN38 links to the complex through a covalent bond, preventing in such a way from the ligature of the DNA strand. As clearly written in the title of [38], SN38 acts like a "foot in the door": it keeps opened the DNA strand to which Top1 is linked as to prevent a door from closing. These complexes are still reversible and do not cause DNA lesions. However, they favor them: some lesions can rise as a consequence of the possible collisions with the transcription complexes or with the replication fork. This induces the arrest of the cell cycle. In this case we speak of irreversible complexes. Lesions due to the inhibition of Top1 are therefore consecutive to the stages of the cell metabolism. It means that irinotecan injections must be repeated and abundant in order to be effective. Besides irinotecan is more effective during the DNA replication phase [36, 46]. Fur-

thermore, the inhibition of the DNA synthesis takes rapidly place (in a few minutes) and lasts several hours.

Defence answers of cells subjected to irinotecan injections are multiple and vary according to the drug dose. The administration of a very light dose suffices to slow down the S phase of the cell cycle and to delay the G2-M transition. If the dose is more substantial, the lag time in the S phase is much more significant and the cell cycle arrest in the G2-M transition can last more than sixty hours or even be permanent. In this latter case, some genes responsible for the cell cycle arrest (as an example, p21) and involved in the aptototic pathway are over-expressed. These genes are activated by p53, and this suggests the intervention of the protein in reply to a DNA damage due to the dissociation of Top1 from DNA [46].

In this work we refer to a pharmacokinetics/pharmacodinamics (PK/PD) model of irinotecan developed by Dimitrio [18] and currently further elaborated by Ballesta [4], that takes aim at representing the action of the drug on the body (pharmacodinamic) and the action of the body on the drug (pharmacokinetic), and thus the drug metabolism and its transformations. This model is made up of 8 differential equations. The following two formulae specify the behaviour of this model.

$\mathbf{F}_{1\_irin}$**:** In case there is no irinotecan, DNA damage equals 0.
LTL$(\mathbb{R})$ **:**$\mathbf{G}([CPT11] = 0) \rightarrow \mathbf{G}([DNAdam] = 0)$.

$\mathbf{F}_{2\_irin}$**:** If the irinotecan concentration is greater than 10, then there exist a future state when DNA damage exceeds the value 0.7 and then stays high.
LTL$(\mathbb{R})$ **:**$\mathbf{G}([CPT11] > 10) \rightarrow \mathbf{FG}([DNAdam] > 3.5)$.

3.5. Irinotecan Exposure Control

In a cancer chronotherapy, an anticancer drug such as irinotecan is injected according to some control law over time. The control law can be represented by a series of parameterized events defining injection times and doses. An event is associated to the beginning and to the end of each injection. Parameters are used to characterize the lapse of time between consecutive injections. The injection control law is part of the system and represented as a component in its own right in the system.

The aim chosen here will be to minimize the toxicity (i.e., DNA damage on healthy cells, that are synchronized) while maintaining a fixed efficacy (the cancer cells lack circadian synchronization and thus efficacy will be supposed constant when the total amount of Irinotecan injected is constant, which is the assumption that the clinicians we collaborated with made). The parameters will thus be those defining a periodic step function with a fixed total area.

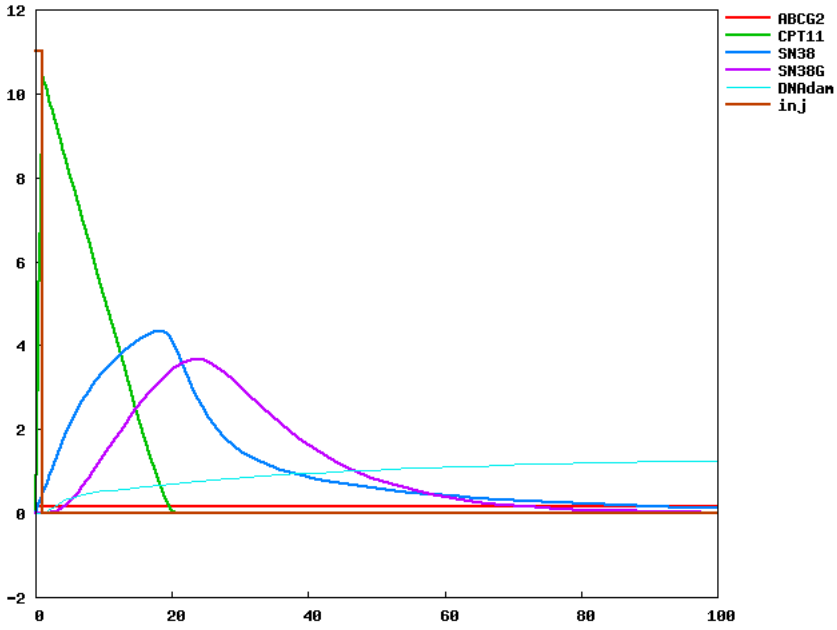


Figure 4: Similuation plot of irinotecan metabolism.

Note that since the model considered only focusses on cellular pharmacokinetics (PK) and pharmacodynamics (PD), and not full-body PKPD, the law to be optimized is the exposure law instead of the injection law. Optimization of the injection law would follow a similar procedure but for a model with a defined target tissue and the corresponding PKPD.

4. Coupled Model Specification

4.1. Model Alignment

The first step of model coupling is model alignment for putting the models in the same format and normalizing molecule names. SBML versions of the irinotecan and p53/Mdm2 modules being available, they were imported in Biocham. The renaming of the variable representing DNA-damage was the only modification necessary in this precise case. More generally it would be necessary to rely on existing databases and ontologies to match corresponding entities in different models.

For the other models, we looked in parallel at the corresponding set of ordinary differential equations and at the available diagrammatic notation to write a set of Biocham reaction rules. Since ODEs can be automatically

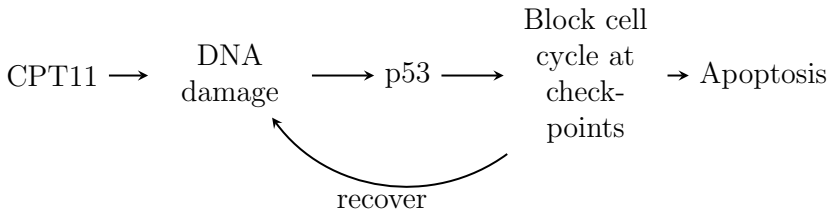


Figure 5: Schematic behaviour of the coupled model.

extracted back from the reactions, one can easily check that the reaction rule models are indeed coherent with the original ODE ones.

4.2. Structural Coupling

The literature provides information about known structural links between the different building blocks to assemble them and compose the coupled model. Before that, let us examine the expected behaviour of the cell which is graphically depicted in Figure 5. Injections of irinotecan (CPT11) induce DNA damage. In reply to this, the cell reacts by activating protein p53, which blocks the cell cycle at a checkpoint. This arrest aims at repairing critical damage before DNA replications occurs, thereby avoiding the propagation of genetic lesions to progeny cells. Thus, while the cell cycle is arrested, the protein p53 will activate the DNA-damage repair mechanisms. If it is possible for the cell to recover, the cell cycle will be restarted; otherwise, if the damage is too extensive, the cell will undergo apoptosis.

As remarked in Section 3, literature provides evidence for the fact that, if a cell is exposed to irinotecan during the S phase of the cell cycle, then more DNA damage will be caused with respect to the other phases of the cell cycle [36, 46]. Keeping this fact in mind, we provided a characterization of the S phase in terms of the concentration level of CycA/Cdk2 (CycA for short) and we inserted in the irinotecan model a dependence from the S phase of the kinetic parameter involved in the production of DNA damage generated by the ternary reversible complexes SN38-Top1-DNA (Top1cc for short): such a parameter assumes a high value during DNA replication and a low value out of synthesis. In this way we linked the cell cycle model to the irinotecan one.

The structure of the coupling of the five models together is illustrated in Figure 6.

The link between the irinotecan model and the p53/Mdm2 one is given by DNA damage. In fact, irinotecan exposure causes DNA damage, which in turn triggers the activity of protein p53, that tries to recover DNA damage.

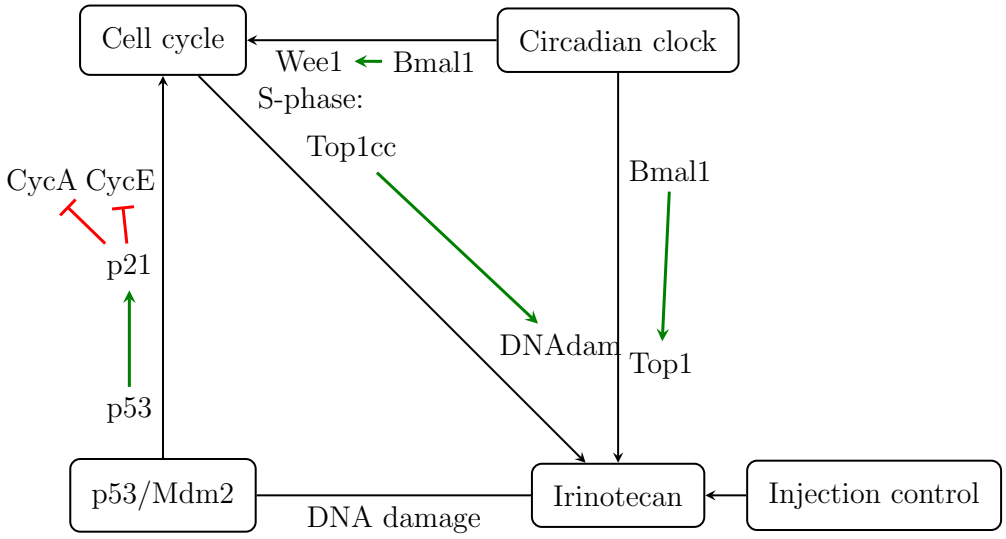


Figure 6: Global schema of the coupled model.

The link between the cell cycle and circadian clock models comes from the experiments of [33] and is reflected through a direct influence of CLOCK-BMAL1 (Bmal1) on the synthesis of Wee1, a kinase that delays or prevents entry into mitosis by phosphorylation of the Cdk1/CyclinB complex. This link uses the same structure as [9] since the Circadian clock model is the same. [45] relied on a slightly different coupling that also modified, for unclear reasons, the reaction of CyclinB synthesis, whereas the aim here is to search for a coupling as simple as possible and satisfying the specification. Note that experimental results direct at a G2/M-transition focussed coupling but that for these experiments the cell-cycle model considered, even if it displays the four different phases, is centered around the restriction point following G1/S.

Bmal1 is also involved in the transcription of Top1 [44]: this provides a link between the circadian clock and irinotecan models.

In order to link the p53/Mdm2 and cell cycle models, we inserted in the p53/Mdm2 model a rule which fixes that p53 activates p21, and two further rules imposing that p21 inhibits CycA and CycE, respectively. It is worth noting that we also investigated the possibility to abstract the previous *expanded* rules by letting p53 directly inhibit CycA and CycE. In the following, we will refer to this last version of the link as to the *contracted* one.

4.3. LTL(ℝ) Specification of the Coupling

In this section, we show how the integration of temporal logic constraints and parameter optimization techniques can be used to compute kinetics for the coupled model. It is worth noticing that for this purpose, one can take advantage of LTL(ℝ) formulae to express numerical constraints in a much more flexible way than by curve fitting, especially for oscillation constraints for instance.

The state transition structure is constituted of a simulation trace over a time window of 100 hours, containing the values of the system's variables of their first derivatives at discrete time points obtained by numerical integration (using Rosenbrock's implicit method for stiff systems).

We directly considered the model made up by the five components and all the linking rules, as illustrated in Figure 6, to perform the parameter research. For the sake of clarity, we will separately introduce each linking rule and the corresponding specification, but as a matter of fact we executed Biocham's parameter optimization procedure only ones to infer the unknown kinetic parameter values leading to the satisfaction of the conjunction of all the formulae given in the following.

The link between the circadian clock and irinotecan models (see Figure 6) has been encoded by means of the following reaction rule, that specifies a mass action law kinetics with parameter `kbmaltop` for the synthesis of Top1:
`MA(kbmaltop) for _=[Bmal1_nucl]=>TOP1.`
The irinotecan model already included the following rule for the synthesis of Top1:
`top1 for _=>TOP1.`
To keep the Top1 production limited and to constrain the correlation between the concentration values of Top1 and Bmal1, suitable values for `top1` and `kbmaltop` such that property F1 holds have been found out.
F1: Top1 is always lower than 1.5 and, whenever Bmal1 gets over 1 (before 85 time units), there exists a future state where Top1 is greater than 1.
LTL(ℝ) : $\mathbf{G}([TOP1] < 1.5 \wedge ([Bmal1\_nucl] > 2.5) \wedge Time < 85 \rightarrow \mathbf{F}([TOP1] > 1))$.
Results: we found out that the values `top1`=0.212 and `kbmaltop`=0.207 make F1 true.

The following Biocham rule encodes the link between the circadian clock and cell cycle models:
`(ksweemp+ksweem*[Bmal1_nucl])/(kweem+kwpcn*[PER_nucl-CRY_nucl])`
`for _=[Bmal1_nucl]=>Wee1.`
While the cell cycle compounds oscillate with a period of approximately 23 hours, the circadian compounds exhibit a period close to 24 hours. To make the cell cycle properly be entrained by the circadian cycle, the cell cycle

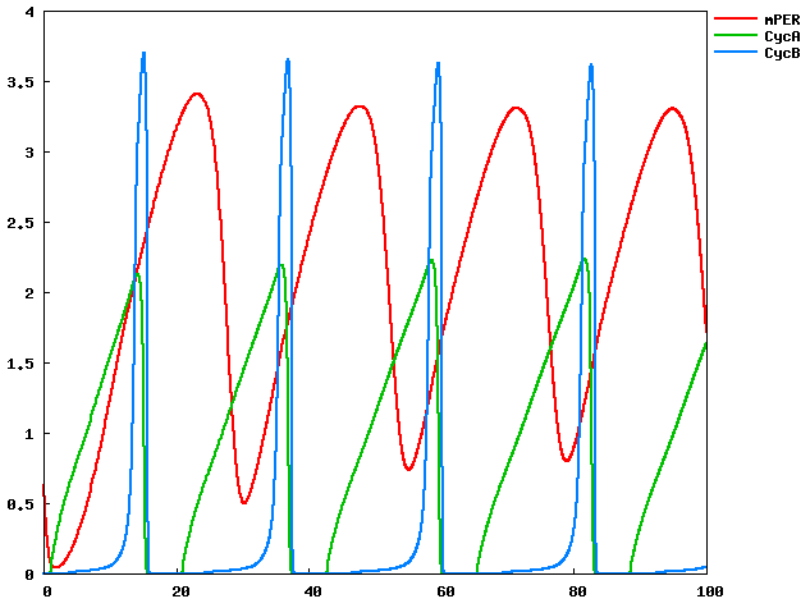


Figure 7: Simulation plot of the cell cycle (CycA, CycB) and circadian clock (mPER) with entrainment knock-out (no coupling). The cell cycle exhibits a free period of 23 hours.

compounds have been required to oscillate with a period of approximately 24 hours, that is, we searched for values for the kinetic parameters involved in the above reaction rule so that property F2 is satisfied (see Figures 7 and 8).

F2: The period of CycA and CycB is 24.

LTL($\mathbb{R}$) : $period(CycA, 24) \wedge period(CycB, 24)$.

Results: the values we found are `ksweemp`=0.521, `ksweem`=0.5, `kweem`=1, and `kwpcn`=2.

Hereafter the Biocham rules introduced to link the p53/Mdm2 and cell cycle models are reported:

```
MA(k5321) for _=[p53]=>p21.
MA(kA21) for CycA=[p21]=>_.
MA(kA21) for CycE=[p21]=>_.
```

As for the contracted version, the encoding is the following one:

```
MA(kA53) for CycA=[p53]=>_.
MA(kA53) for CycE=[p53]=>_.
```

Again, suitable parameter values for `k5321` and `kA21` (`kA53` in the second case) have been searched so that property F3, that expresses the CycA oscillating behaviour exhibited by the cell cycle model when entrained by the

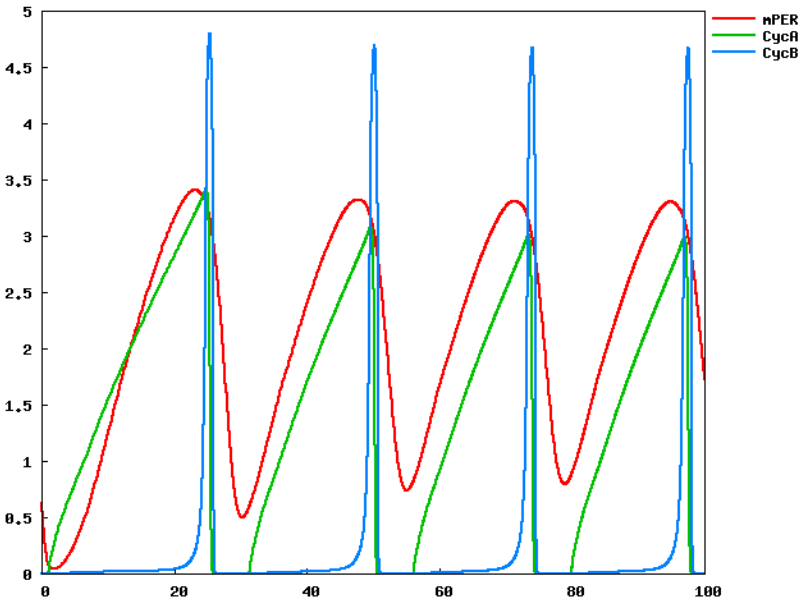


Figure 8: Simulation plot of the entrainment of the cell cycle by the circadian clock through coupling on Wee1. The period of the cell cycle is 24 hours.

circadian clock, is conserved when the p53/Mdm2 module is added but there is no irinotecan exposure.

F3: Within a time interval of 100 time units, CycA is greater than 2.7 in at least 4 oscillations.

LTL($\mathbb{R}$) : $oscil([CycA], 4, 2.7)$.

Results: suitable parameter values are `k5321=0.487`, `kA21=0.00507`, and `kA53=0.283`. Property F3 also turned out to be true when there is exposure to irinotecan but the p53/Mdm2 model is not taken into account. In fact, as expected, even if DNA damage occurs, when protein p53 does not act, the cell cycle is not affected, and thus CycA exhibits a regular oscillating behaviour.

Finally, to link the cell cycle and irinotecan models, the following rule has been considered

```
MA(kdam) for TOP1cc=>DNAdam.
```

already included in the irinotecan model and we made the parameter `kdam` depend from the S phase: it assumes a first value $v1$ during replication, a second value $v2$ out of replication, where $v2 > v1$. We searched for suitable values $v1$ and $v2$ for `kdam` so that the next property holds.

F4: Whenever Top1cc gets above 0.2, there exists a future state when the

first derivative of DNAdam gets above 0.15.
LTL($\mathbb{R}$) : $\mathbf{G}([TOP1cc] > 0.2 - > \mathbf{F}(d([DNAdam])/dt > 0.15))$.
Results: the property is verified for $v1$ equal to 1.42 and $v2$ equal to 1.89.

The combination of temporal logic constraints and parameter optimization techniques can also be used to validate the resulting model. As an example, the next property ascertains that, in case of repeated irinotecan exposure (and thus of sustained DNA-damage) the oscillations of CycA are affected.
F5: When there is sustained DNA damage (after an initial period), the amplitude of CycA decreases before 73 time units and then stays low.
LTL($\mathbb{R}$) : $\mathbf{F}((Time < 15) \wedge \mathbf{G}([DNAdam] > 0.4)) \rightarrow \mathbf{F}((Time < 73) \wedge \mathbf{G}([CycA] < 2.15))$.
Results: with the expanded version of the links the amplitude of oscillations gradually decreases, satisfying the property. With the contracted one, oscillations are very irregular, as graphically depicted in Figure 9 (bottom panel).

5. Evaluation of the Parameter Search Procedure

The method used in Biocham to optimize parameter values with respect to LTL($\mathbb{R}$) properties consists in computing a continous satisfaction degree in $[0, 1]$ for a temporal logic formula on a given simulation trace [41], using an algorithm for computing validity domains of LTL($\mathbb{R}$) constraints with free variables instead of constants [22, 21] and then using the continuous satisfaction degree as fitness function for a continous optimization method.

Biocham uses the state-of-the-art nonlinear optimization method of Hansen and Ostermeier [27] named Covariance Matrix Adaptation Evolution Strategy (CMA-ES). A population of new candidate solutions is sampled according to a multivariate normal distribution of the parameters. The covariance matrix adaptation is a method to update the covariance matrix of this distribution. This method is a generalization of the approximate gradient and Hessian of a quasi-Newton method to an evolutionary algorithm for optimization problems with a black box fitness function on which no assumption is made. CMA-ES performs parameter search given an initial solution, stop and restart criteria, and a given search space. The search stops either when a given number of violation degrees have been computed or when the violation degree gets below a given threshold.

We searched for parameter values satisfying all F1 to F5 properties, each property being evaluated for a given set of models. The overall fitness of a parameter values set is the sum of the fitness of all these properties. This computation is done in parallel as well as the computation of the fitness of

Parameter	Value	Formula
kbmaltop	0.207	F1
top1	0.212	F1
ksweemp	0.521	F2
ksweem	0.5	F2
kweem	1.12	F2
kwpcn	5	F2
k5321	0.486	F3
kA21	0.00507	F3
kA53	0.283	F3
v1	1.42	F4
v2	1.89	F4

Table 1: Parameter values learned in Biocham, values found, and temporal logic formulae used for learning them.

the population of solutions defined by CMA-ES. One 100h simulation of the complete model takes about 100s on a 3GHz processor and thus the evaluation of the 5 properties combined can take up to 500s. It took around 1000 evaluations of these five properties combined to find a satisfactory solution. The execution time was 4 hours on 64 3GHz cores.

Such a temporal logic constraint approach proved to be effective, allowing us to express relevant biological properties of the model (and concentration values that make specifications true) that could not be easily encoded as curve fitting problems for instance. This is the case in Figures 7 and 8 which depict the behavior of the cell cycle when it is respectively entrained and not entrained by the circadian clock. While in the first case a period of approximately 23 hours is exhibited by CycA and CycB, in the second one the two compounds assume the same period of the circadian cycle, that is, approximately 24 hours. In Figure 7, the disruption of the first oscillation is due to the knock out of the entrainment reaction.

The set of the linking parameter values learned in Biocham with this procedure, together with the temporal logic formulae used for learning them, are recapitulated in Table 1

6. Optimal Control of Drug Exposure

The properties of this subsection deal with the control laws of irinotecan exposure. In order to deal with chronotherapeutics optimization for healthy cells while maximizing efficacy for tumor cells, we aimed at finding irinotecan

exposure times and maximum amount that maintain toxicity low for healthy cells. More precisely, we modeled irinotecan exposure as rectangular boxes and we looked for maximum irinotecan quantity and for first exposure time, interval time between consecutive exposure and boxes width and height that keep DNA damage below a given threshold.

6.1. Evaluation of pulsatile exposure

In Figure 9 we show the behavior of the p53/Mdm2 DNA damage repair module when exposure is repeated every 24 hours. The plot puts in evidence how DNA damage increases after every exposure period. The oscillating trend of proteins p53 and Mdm2 is well highlighted. Furthermore, it is possible to notice the irregular behaviour assumed by CycA after exposure to irinotecan if the contracted link is used (bottom panel).

6.2. Optimization of the drug exposure law

To find the most efficient exposure law, we searched for the optimal schedule and the maximum amount of irinotecan such that DNA damage remains below a given threshold.

F6: DNAdam is always lower than 1 and total irinotecan exposure is greater than 50.
LTL($\mathbb{R}$) :G$([DNAdam] < 1) \wedge totalinjection > 50$.
We searched for parameter values that make F6 be satisfied with the lowest error, i.e., the values that maintain DNA damage below 1 and that minimize the distance between total irinotecan exposure and value 50. To avoid too short injections, the minimal injection length has been set to 1.
Results: the maximum irinotecan exposure maintaining DNA damage low, are rectangular boxes with a width 1 (e.g., the lower bound we set to injection length) and a height of approximately 7. The first exposure should happen 23h30 after the initial state (chosen with CycA very low, i.e., in G1 phase), and a new exposure every cell cycle oscillation should then be done (see Figure 10, where the macro KDAM delineates the synthesis phase of the cell cycle).

It is worth noting that the formation of the ternary reversible complexes SN38-Top1-DNA (Top1cc) responsible for DNA damage follows the irinotecan exposure by a few hours and that the choice of irinotecan exposure described above corresponds at having Top1cc peaks out of the synthesis phase (remember that the production of DNA damage from Top1cc is higher during the S phase). On the other hand, the presence of Top1cc peaks during the replication phase leads at maximizing toxicity. Figure 11 shows the effect of the same exposure than for Figure 10 but with a 12h phase shift. With this

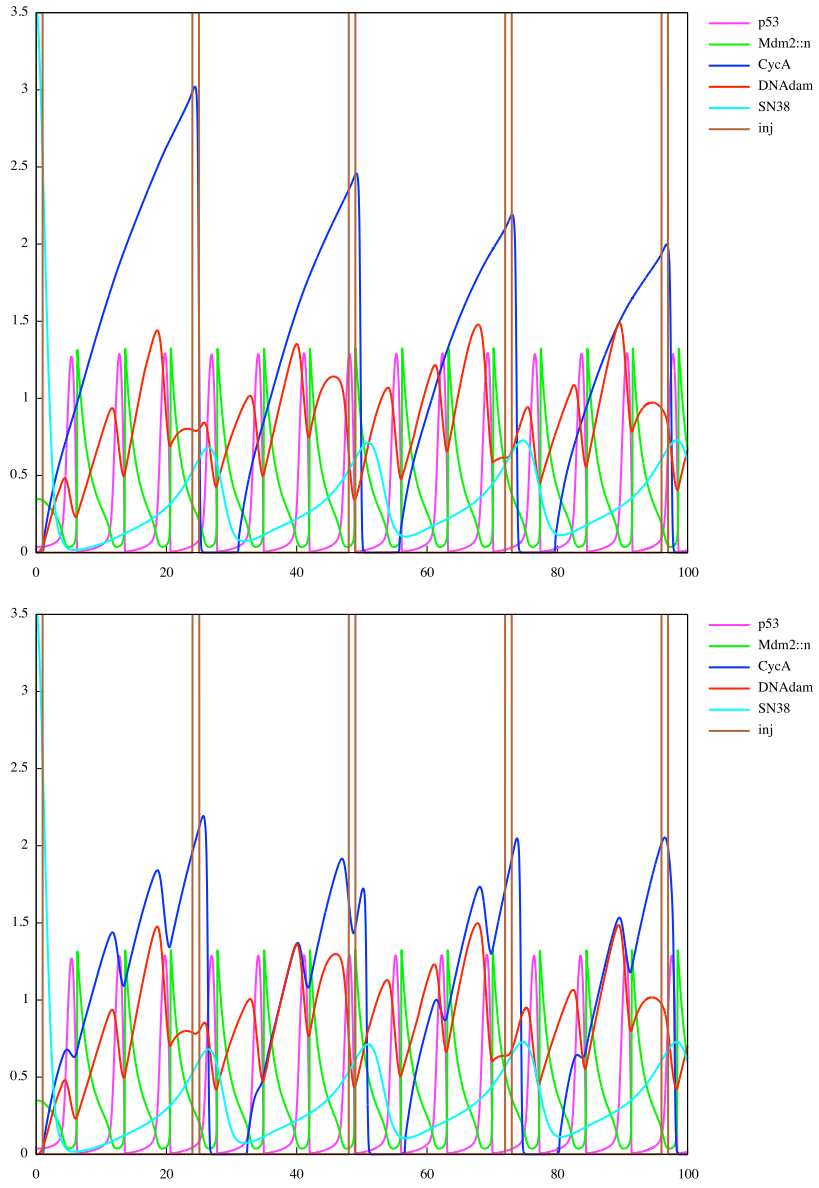


Figure 9: Simulation plot of DNA damage under pulsatile exposure to irinotecan every 24 hours with the p53/Mdm2 module. In the bottom panel, the contracted link is used, which results in very irregular CycA oscillations.

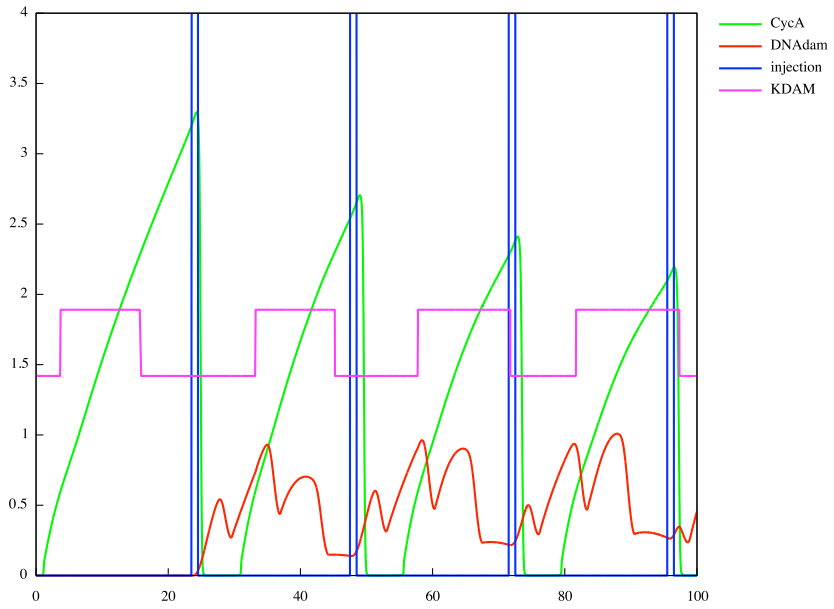


Figure 10: Maximum exposure preserving DNA damage under threshold 1.

phase shift, which can be attained for unsynchronized cells, DNA damage attains 1.7, that is a 70 percent increase compared to synchronized cells.

The next specification regards the DNA repairing power of the cell.

F7: After an exposure to irinotecan is performed, DNA damage is able to go under the threshold of 0.1 before the next exposure.

LTL($\mathbb{R}$) : $\mathbf{G}(([CPT11] > d) \vee (([CPT11] \leq d)\mathbf{U}([DNAdam] < 0.1)))$., where d depends on the dose of irinotecan.

Before testing the property, we decided to parameterize the lapse of time between consecutive irinotecan exposures. Then we took advantage of the procedure `learn_parameters` to find the minimum k such that, if one 10-units-exposure is performed every k hours, then property F7 is true.

Results: we found out that the minimum k multiple of 12 which makes F7 true is 36. Thus, one exposure every 36 hours should be performed in order to allow DNA damage to be recovered before the next exposure. Then we tried to see what it happens if, at each exposure, we double the irinotecan dose, that is, we expose to 20 units. In this case, one exposure every 48 hours should be done.

The last property requires the oscillating trend of proteins p53 and Mdm2 to stop before a new exposure.

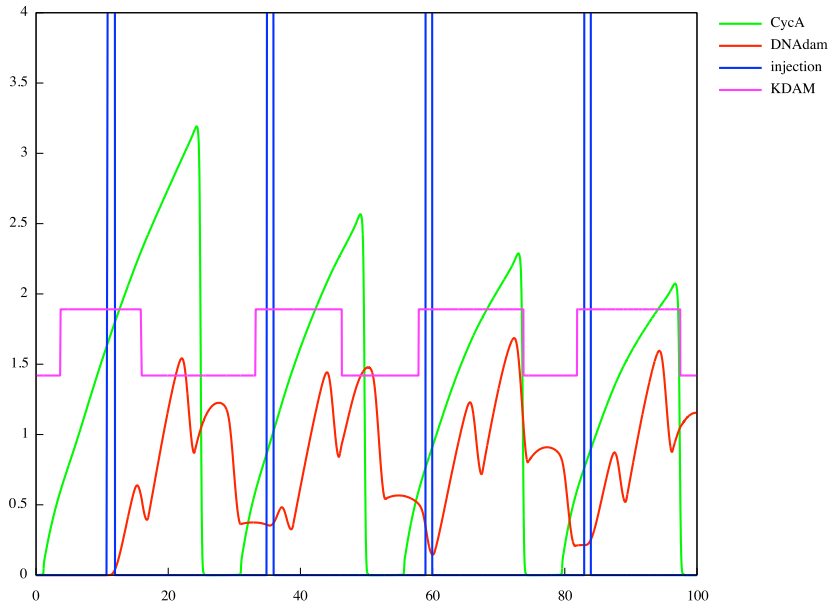


Figure 11: DNA damage produced on phase-shifted cells with the same exposure law as in Figure 10.

F8: When exposed to irinotecan, p53 and Mdm2 are in a steady state, that is, their derivatives approach 0.
LTL($\mathbb{R}$) : $\mathbf{G}((([CPT11] > d) \rightarrow ((d[p53] \leq 0.05) \wedge (d[p53] \geq -0.05)$
$\wedge (d[Mdm2 :: n] \leq 0.05) \wedge (d[Mdm2 :: n] \geq -0.05)))$., where d depends on the dose of irinotecan.
As for the previous specification, we parameterized the lapse of time between consecutive irinotecan exposures and we used the procedure `learn_parameters`.
Results: the minimum k multiple of 12 which makes F8 true is 48.

7. Model Predictions for Circadian Clock Genes Knock-outs

7.1. Setup

Hereafter we describe how the cell cycle reacts to circadian gene/protein mutations in our coupled model.

This really amounts to verifying the predictive power of the model since, as already explained, the circadian entrainment is focussed on the G2/M transition, whereas the cell cycle model is focussed on the restriction point.

We explore what happens when a given compound is missing, that is, its concentration equals zero. To this aim, it is possible either to set the compound synthesis at zero, or to make the compound be absorbed by a "super-inhibitor" (e.g., the knock-out of a given compound C can be modeled by inserting in the model the rule $Inhibitor + C \rightarrow Inhibitor\text{-}C$, where the initial concentration of $Inhibitor$ is very high). As a matter of fact, both the alternatives have the same impact on the behavior of the coupled model. The mutations we take into consideration concern the mRNAs *mPER*, *mCRY*, and *mBmal1*. The simulations we provide in the following are up to 100 hours.

mPER=0. As shown in Figure 12, in this case the cell cycle period becomes bigger (approximately 28.5 hours), that is, the cellcycle is slowed down. Furhermore, the mean value of Wee1 is higher with respect to the one of the wild type phenotype.

mCRY=0. In this case the behavior of the model is approximately the same of the previous one (see Figure 12).

mBmal1=0. As illustrated in Figure 13 in this mutant the mean value of Wee1 is lower with respect to normal conditions and the cell cycle period is slightly smaller (approximately 23 hours).

7.2. Comparison with the Literature

In the following we itemize some facts we found in literature concerning the dependence of the cell division cycle on circadian rhythmicity/mutations

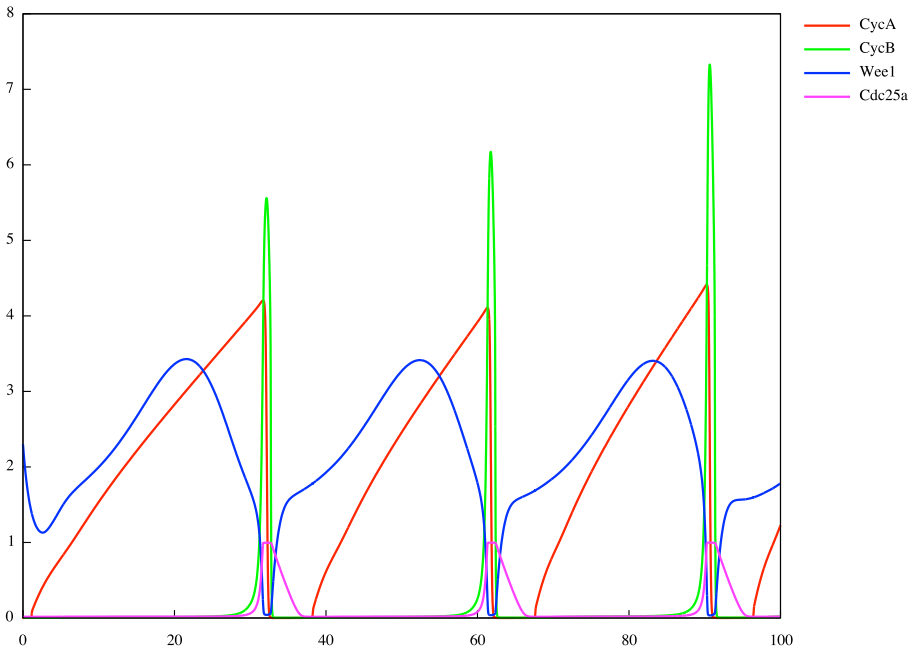


Figure 12: Simulation of the coupled model with *mPER=0*.

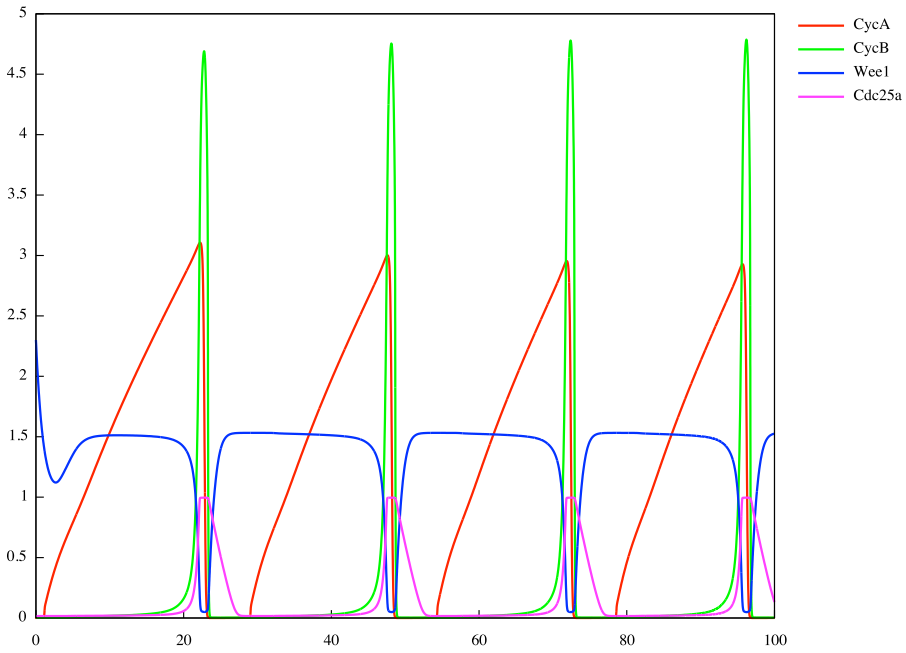


Figure 13: Simulation of the coupled model with *mBmal1=0*.

and, when possible, we discuss the consistency with our results.

- The expression of several mammalian cell-cycle genes, including *c-myc*, *Cyclin-D1*, and *mWee1*, is regulated in a circadian manner [30]. As expected we also observe the circadian entrainment of cell-cycle genes in our *in silico* model.

- Overexpression of *PER1* leads to apoptosis whereas inhibition of *PER1* inhibits apoptosis. It appears that *PER1* antagonizes the cell cycle in an oscillatory fashion similar to the manner in which it antagonizes the function of Clock-Bmal1 [30]. According to our experiments, a *PER* inhibition produces an increase of *Wee1*, an thus a mitosis inhibition. Note however that we observe a lengthened period and not a complete stop of the mitosis.

- In *CRY* deficient cells, the circadian rhythmicity is lost [43], Wee1, over-expressed, and CyclinB, less active, loses rhythmicity [33]. The effect on *Wee1* and *CyclinB* is roughly consistent with our results.

There are also some KOs that were not directly comparable with our results since our model does not incorporate yet detailed DNA-damage pathways with ATR/ATM, Chk1/2 or cMyc:

- *PER1* and *TIM* seem implicated on the DNA-damage response because both can be found complexed with the *ATM* and *ATR* kinases and the checkpoint kinases *Chk2* and *Chk1*, respectively [30].

- The oscillatory expression of *c-myc* is abolished in *mPER2* mutant mice, which could then result in an alteration of the *p53* function [30].

7.3. Gene KOs Conclusion

We observe that for most of the knock-outs, the results of our coupled model are in accordance with experimental data, which considering the very simple specification used for the coupling is a quite interesting result.

For other mutations that should result in a complete stop of the mitosis, the result does not agree with the data since our model exhibits a slow down of the cell cycle but not an arrest in mitosis. This points out a weakness of the mammalian cell cycle model we have used. It is indeed driven by a constantly growing mass variable and focusses on the restriction point with few details on the G2/M transition. While this control of the mitosis by the mass variable is realistic in yeast, it limits the possibility of controlling the cell cycle in mammalian cells, it is thus virtually impossible to block at

the corresponding checkpoint, even with a strong circadian coupling. These considerations motivate the use of cell cycle control models independent of the mass variable [25, 39] allowing for more accurate predictions in this respect [16].

This evaluation of the model predictions on gene knock-outs also show that model-checking and parameter search are useful at the prediction stage: not finding any satisfactory parameter set when trying to strengthen the coupling in order to agree with the experimental result, indeed reveals a weakness in the structure of the individual models, which needs be revised in order to make the specification satisfiable.

8. Conclusion and Perspectives

In this paper, we have presented a coupled model of the mammalian cell cycle, circadian clock, p53-based DNA-damage repair, irinotecan intracellular PK/PD, and irinotecan exposure control, in order to study the influence of irinotecan drug in cancer chronotherapies. The coupling of the composite models has been achieved in Biocham using an original method based on LTL($\mathbb{R}$) temporal logic constraint solving, for representing the expected behavior of the coupled system, and on a continuous optimization evolutionary algorithm for inferring the values of the unknown coupling kinetic parameters of the models, as well as the exposure control parameters.

The maximization of antitumor effects and the minimization of the toxicity on healthy cells is the aim of any cancer therapy. The rationale of irinotecan chronotherapies is its toxicity on the cells in S phase only, the synchronization of the cell cycle by the circadian clock in healthy tissue cells, and the circadian disruption in mutated cancer cells. The resulting coupled model provides a valuable tool to investigate the drug influence on the cell cycle, reveal some weaknesses in the models, and ultimately infer some properties concerning the drug therapy and optimal exposure times and doses.

The predictive power of the coupled model was tested with respect to a limited set of mutants of the circadian clock genes. In the case of genes knock outs, we succeeded in considering temporal logic constraints over different traces corresponding to the mutations of different genes, that is, the initial condition of the trace relative to the knock out of a given set of genes is characterized by setting at 0 the parameters involved in the synthesis of the genes.

Although preliminary, the results obtained are very encouraging for our coupling method. In particular they showed that mass-entrained models of the cell-cycle have a limited possibility of entrainment by the circadian molecular clock. This motivates the use of non mass entrained cell cycle

models like [25, 39, 16] which should not suffer from this limitation. The results also showed that the p53-Mdm2 DNA damage repair model of [13] should be improved in order to introduce a threshold above which the DNA is no longer repaired and the cell enters apoptosis. Last but not least, a PK/PD model of irinotecan in the body is missing to link the irinotecan injection law to the cell exposure model and optimize the drug injection law directly.

Acknowledgements

This work was supported by the EU FP6 STREP project TEMPO on cancer chronotherapies and is now supported by the ERASysBio project C5Sys concerning circadian and cell cycle clock systems in cancer. We acknowledge fruitful discussions with the partners of this project, in particular with Francis Lévi, Jean Clairambault and Annabelle Ballesta.

[1] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell, fourth edition.* Garland Science, 2008.

[2] A. Altinok, F. Lévi, and A. Goldbeter. A cell cycle automaton model for probing circadian patterns of anticancer drug delivery. *Advanced Drug Delivery Reviews*, 59:1036–1053, 2007.

[3] Marco Antoniotti, Alberto Policriti, Nadia Ugel, and Bud Mishra. Model building and model checking for biochemical processes. *Cell Biochemistry and Biophysics*, 38:271–286, 2003.

[4] A. Ballesta, S. Dulong, C. Abbara, B. Cohen, A. Okyar, F. Levi, and J. Clairambault. A combined biological and mathematical study of the anticancer drug irinotecan molecular pharmacokinetics-pharmacodynamics and their control by the circadian clock. In preparation.

[5] G. Batt, C. Belta, and R. Weiss. Temporal logic analysis of gene networks under parameter uncertainty. *IEEE Transactions on Circuits and Systems and IEEE Transactions on Automatic Control*, 58(Joint Special Issue on Systems Biology):215–229, 2008.

[6] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004. Applications note.

[7] Laurence Calzone, Nathalie Chabrier-Rivier, François Fages, and Sylvain Soliman. Machine learning biochemical networks from temporal logic properties. In Gordon Plotkin, editor, *Transactions on Computational Systems Biology VI*, volume 4220 of *Lecture Notes in BioInformatics*, pages 68–94. Springer-Verlag, November 2006. CMSB'05 Special Issue.

[8] Laurence Calzone, François Fages, and Sylvain Soliman. BIOCHAM: An environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807, 2006.

[9] Laurence Calzone and Sylvain Soliman. Coupling the cell cycle and the circadian cycle. Research Report 5835, INRIA, February 2006.

[10] Nathalie Chabrier and François Fages. Symbolic model checking of biochemical networks. In Corrado Priami, editor, *CMSB'03: Proceedings of the first workshop on Computational Methods in Systems Biology*, volume 2602 of *Lecture Notes in Computer Science*, pages 149–162, Rovereto, Italy, March 2003. Springer-Verlag.

[11] Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, François Fages, and Vincent Schächter. Modeling and querying biochemical interaction networks. *Theoretical Computer Science*, 325(1):25–44, September 2004.

[12] Vijay Chickarmane, Animesh Ray, Herbert M. Sauro, and Ali Nadim. A model for p53 dynamics triggered by dna damage. *SIAM Journal on Applied Dynamical Systems*, 6:61–78, 2007.

[13] Andrea Ciliberto, Béla Novák, and John J. Tyson. Steady states and oscillations in the p53/mdm2 network. *Cell Cycle*, 4(3):488–493, March 2005.

[14] Edmund M. Clarke, James R. Faeder, Christopher James Langmead, Leonard A. Harris, Sumit Kumar Jha, and Axel Legay. Statistical model checking in biolab: Applications to the automated analysis of t-cell receptor signaling pathway. In Monika Heiner and Adeline Uhrmacher, editors, *CMSB'08: Proceedings of the fourth international conference on Computational Methods in Systems Biology*, volume 5307 of *Lecture Notes in Computer Science*, pages 231–250. Springer-Verlag, October 2008.

[15] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[16] Elisabetta De Maria, François Fages, and Sylvain Soliman. Model-based predictions of the influence of circadian clock genes knock-outs on the cell cycle. INRIA Research Report RR-7064, INRIA, June 2009.

[17] Elisabetta De Maria, François Fages, and Sylvain Soliman. On coupling models using model-checking: Effects of irinotecan injections on the mammalian cell cycle. In *CMSB'09: Proceedings of the seventh international conference on Computational Methods in Systems Biology*, volume 5688 of *Lecture Notes in BioInformatics*, pages 142–157. Springer-Verlag, 2009.

[18] Luna Dimitrio. Irinotecan: Modelling intracellular pharmacokinetics and pharmacodynamics. m2 master thesis (in french, english summary). Technical report, University Pierre-et-Marie-Curie and INRIA internal report, 2007.

[19] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, José Meseguer, and M. Kemal Sönmez. Pathway logic: Symbolic analysis of biological signaling. In *Proceedings of the seventh Pacific Symposium on Biocomputing*, pages 400–412, January 2002.

[20] François Fages. Temporal logic constraints in the biochemical abstract machine BIOCHAM (invited talk). In Springer-Verlag, editor, *Proceedings of Logic Based Program Synthesis and Transformation, LOPSTR'05*, number 3901 in Lecture Notes in Computer Science, London, UK, September 2005.

[21] François Fages and Aurélien Rizk. On temporal logic constraint solving for the analysis of numerical data time series. *Theoretical Computer Science*, 408(1):55–65, November 2008.

[22] François Fages and Aurélien Rizk. From model-checking to temporal logic constraint solving. In *Proceedings of CP'2009, 15th International Conference on Principles and Practice of Constraint Programming*, number 5732 in Lecture Notes in Computer Science, pages 319–334. Springer-Verlag, September 2009.

[23] François Fages, Sylvain Soliman, and Aurélien Rizk. *BIOCHAM v2.8 user's manual*. INRIA, 2009. `http://contraintes.inria.fr/BIOCHAM`.

[24] Daniel B. Forger and Charles S. Peskin. A detailed predictive model of the mammalian circadian clock. *Proceedings of the National Academy*

of Sciences of the United States of America, 100(25):14806–14811, December 2003.

[25] Claude Gérard and Albert Goldbeter. Temporal self-organization of the cyclin/cdk network driving the mammalian cell cycle. *Proceedings of the National Academy of Sciences*, 106(51):21643–21648, December 2009.

[26] N. Geva-Zatorsky, N. Rosenfeld, S. Itzkovitz, R. Milo, A. Sigal, E. Dekel, T. Yarnitzky, Y. Liton, P. Polak, G. Lahav, and U. Alon. Oscillations and variability in the p53 system. *Molecular Systems Biology*, 2, 2006.

[27] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[28] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. In *Proc. Computational Methods in Systems Biology (CMSB'06)*, volume 4210 of *Lecture Notes in Computer Science*, pages 32–47. Springer-Verlag, 2006.

[29] Michael Hucka et al. The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[30] Tim Hunt and Paolo Sassone-Corsi. Riding tandem: Circadian clocks and the cell cycle. *Cell*, 129(3):461–464, May 2007.

[31] Kurt W. Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Molecular Biology of the Cell*, 10(8):2703–2734, August 1999.

[32] Jean-Christophe Leloup and Albert Goldbeter. Toward a detailed computational model for the mammalian circadian clock. *Proceedings of the National Academy of Sciences*, 100:7051–7056, 2003.

[33] Takuya Matsuo, Shun Yamaguchi, Shigeru Mitsui, Aki Emi, Fukuko Shimoda, and Hitoshi Okamura. Control mechanism of the circadian clock for timing of cell division in vivo. *Science*, 302(5643):255–259, October 2003.

[34] J. Nitiss and J. C. Wang. Dna topoisomerase-targeting antitumor drugs can be studied in yeast. *Proceedings of the National Academy of Sciences of the United States of America*, 85(20):7501–7505, October 1988.

[35] Béla Novák and John J. Tyson. A model for restriction point control of the mammalian cell cycle. *Journal of Theoretical Biology*, 230:1383–1388, 2004.

[36] Shigehiro Ohdo, Tomoko Makinosumi, Takashi Ishizaki, Eiji Yukawa, Shun Higuchi, Shigeyuki Nakano, and Nobuya Ogawa. Cell cycle-dependent chronotoxicity of irinotecan hydrochloride in mice. *Journal of Pharmacology and Experimental Terapeutics*, 283(3):563–579, December 1997.

[37] C. Piazza, M. Antoniotti, V. Mysore, A. Policriti, F. Winkler, and B. Mishra. Algorithmic algebraic model checking i: Challenges from systems biology. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification*, volume 3576, chapter 3, pages 5–19. Springer-Verlag, Berlin, Heidelberg, 2005.

[38] Y. Pommier. Camptothecins and topoisomerase i: A foot in the door. targeting the genome beyond topoisomerase i with camptothecins and novel anticancer drugs: Importance of dna replication, repair and cell cycle checkpoints. *Current Medicinal Chemestry Anticancer Agents*, 4(5):429–434, 2004.

[39] Zhilin Qu, W. Robb MacLellan, and James N. Weiss. Dynamics of the cell cycle: checkpoints, sizers, and timers. *Biophysics Journal*, 85(6):3600–3611, 2003.

[40] Aviv Regev, William Silverman, and Ehud Y. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Proceedings of the sixth Pacific Symposium of Biocomputing*, pages 459–470, 2001.

[41] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In Monika Heiner and Adeline Uhrmacher, editors, *CMSB'08: Proceedings of the fourth international conference on Computational Methods in Systems Biology*, volume 5307 of *Lecture Notes in Computer Science*, pages 251–268. Springer-Verlag, October 2008.

[42] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. A general computational method for robustness analysis with applications to synthetic gene networks. *Bioinformatics*, 12(25):il69–il78, June 2009.

[43] Gijsbertus T. J. van der Horst, Manja Muijtjens, Kumiko Kobayashi, Riya Takano, Shin ichiro Kanno, Masashi Takao, Jan de Wit, Anton Verkerk, Andre P. M. Eker, Dik van Leenen, Ruud Buijs, Dirk Bootsma, Jan H. J. Hoeijmakers, and Akira Yasui. Mammalian cry1 and cry2 are essential for maintenance of circadian rhythms. *Nature*, 398(6728):627–630, April 1999.

[44] F. Yang, Y. Nakajima, M. Kumagai, Y. Ohmiya, , and M. Ikeda. The molecular mechanism regulating the autonomous circadian expression of topoisomerase i in nih3t3 cells. *Biochemical and Biophysical Research Communications*, 380(1):22–27, 2009.

[45] J. Zámborszky, C.I. Hong, and A.C. Nagy. Computational analysis of mammalian cell division gated by a circadian clock: Quantized cell cycles and cell size. *Journal of Biological Rhythms*, 22(6):542–553, 2007.

[46] Y. Zhou, F.G. Gwadry, W.C. Reinhold, L.H. Smith L.D. Miller, U. Scherf, E.T. Liu, K.W. kohn, Y. Pommier, and J.N. Weinstein. Transcriptional regulation of mitotic genes by camptothecin-induced dna damage : Microarray analysis of doseand time-dependent effects. *Cancer Research*, 62:1668–1695, 2002.

9. Annex

In the following we give the reaction rules for the five models. The parameter values and initial conditions are omitted. The complete models can be retrieved at
`http://contraintes.inria.fr/supplementary_material/TCS-CMSB09/`.

9.1. Mammalian Cell Division Cycle Control

```
epsilon*k15/(1+([DRG]/J15)^2)                  for _=>ERG.
MA(k16)                                        for ERG=>_.
MA(epsilon *k17p)                              for _=[ERG]=>DRG.
epsilon*k17*([DRG]/J17)^2/(1+([DRG]/J17)^2)    for _=[DRG]=>DRG.
MA(k18)                                        for DRG=>_.
MA(epsilon*k9)                                 for _=[DRG]=>CycD.
MA(V6)                                         for _=[CycD-Kip1]=>CycD.
MA(k10)                                        for CycD=>_.
(MA(k24),MA(k24r))                             for CycD+Kip1<=>CycD-Kip1.
MA(V6+k10)                                     for CycD-Kip1=>_.
epsilon*(k7p+k7*E2F_A)                         for _=>CycE.
MA(V8)                                         for CycE=>_.
```

```
MA(V6)                                          for _=[CycE-Kip1]=>CycE.
(MA(k25),MA(k25r))                              for CycE+Kip1<=>CycE-Kip1.
MA(V6+V8)                                       for CycE-Kip1=>_.
epsilon*k29*E2F_A*mass                          for _=[massT]=>CycA.
MA(k30)                                         for CycA=[Cdc20]=>_.
MA(V6)                                          for _=[CycA-Kip1]=>CycA.
(MA(k25),MA(k25r))                              for CycA+Kip1<=>CycA-Kip1.
MA(V6)                                          for CycA-Kip1=>_.
MA(k30)                                         for CycA-Kip1=[Cdc20]=>_.
epsilon*k5                                      for _=>Kip1.
MA(V6)                                          for Kip1=>_.
MA(k10)                                         for _=[CycD-Kip1]=>Kip1.
MA(V8)                                          for _=[CycE-Kip1]=>Kip1.
MA(k30)                                         for Cdc20+CycA-Kip1=>Kip1+Cdc20+CycA-Kip1.
k22*E2F_T                                       for _=>E2F.
MA(k22+k23p)                                    for E2F=>_.
MA(k23)                                         for E2F=[CycA]=>_.
MA(k23)                                         for E2F=[CycB]=>_.
epsilon*k1p                                     for _=>CycB.
epsilon*k1*([CycB]/J1)^2/(1+([CycB]/J1)^2)      for _=[CycB]=>CycB.
MA(V2)                                          for CycB=>_.
(k3p+k3*[Cdc20])*(1-[Cdh1])/(J3+1-[Cdh1])       for _=[Cdc20]=>Cdh1.
V4*[Cdh1]/(J4+[Cdh1])                           for Cdh1=>_.
epsilon*k11p                                    for _=>Cdc20_T.
MA(epsilon*k11)                                 for _=[CycB]=>Cdc20_T.
MA(k12)                                         for Cdc20_T=>_.
k13*[IEP]*([Cdc20_T]-[Cdc20])/(J13+[Cdc20_T]-[Cdc20]) for _=[IEP]=>Cdc20.
(k14/(J14+[Cdc20])+k12)*[Cdc20]                 for Cdc20=>_.
epsilon*k33                                     for _=>PPX.
MA(k34)                                         for PPX=>_.
k31*[CycB]*(1-[IEP])/(J31+1-[IEP])              for _=[CycB]=>IEP.
k32*[PPX]*[IEP]/(J32+[IEP])                     for IEP=[PPX]=>_.
k27*mass*(if Rb_hypo/Rb_T >0.8 then 0 else 1)   for _=[massT]=>GMT.
MA(k28)                                         for GMT=>_.
epsilon*mu*nbcells*[GMT]                        for _=[GMT]=>massT.

% Steady-state relations
macro(PP1_A, PP1_T/(1+K21*(Phi_E*([CycE]+[CycA])+Phi_B*[CycB]))).
macro(Rb_hypo, Rb_T/(1+(k20*(lambda_D*CycD_T+lambda_E*[CycE]+lambda_A*[CycA]+lambda_B*[CycB]))
macro(E2F_A,(E2F_T - E2FRb)*[E2F]/E2F_T).
macro(E2FRb, 2*E2F_T*Rb_hypo/(E2F_T+Rb_hypo+L+((E2F_T+Rb_hypo+L)^2 - 4*E2F_T*Rb_hypo)^(1/2))).

% Definitions
macro(V2, k2p*(1 - [Cdh1])+k2*[Cdh1]+k2s*[Cdc20]).
macro(V4, k4*(gamma_A*[CycA]+gamma_B*[CycB]+gamma_E*[CycE])).
macro(V6, k6p+k6*(eta_E*[CycE]+eta_A*[CycA]+eta_B*[CycB])).
macro(V8, k8p+(k8*(Psi_E*([CycE]+[CycA])+Psi_B*[CycB]))/(J8+CycE_T)).
macro(L, k26r/k26+k20/k26*(lambda_D*CycD_T+lambda_E*[CycE]+lambda_A*[CycA]+lambda_B*[CycB])).
```

```
add_event([Cdh1]>0.2,nbcells, nbcells*2).

macro(mass, [massT]/nbcells).
macro(GM, [GMT]/nbcells).

% Make CycB synthesis proportional to mass
delete_rules(_ => CycB).
epsilon*k1p*mass                                  for _ => CycB.

delete_rules( _=[CycB]=>CycB).
epsilon*k1*([CycB]/J1)^2/(1+([CycB]/J1)^2)*mass for _ =[CycB]=> CycB.

% Change the cell division trigger
delete_event([Cdh1]>0.2,nbcells,nbcells*2).
add_event([CycB]<0.2,nbcells,nbcells*2).

% Add Wee1/Cdc25 machinery
macro(V2, k2p*(1 - [Cdh1])+k2*[Cdh1]+k2s*[Cdc20]).

MA(kwee1p)                                 for CycB => CycB~{p}.
MA(kwee1s)                                 for CycB =[Wee1]=> CycB~{p}.
MA(kcdc25p)                                for CycB~{p} => CycB.
MA(kcdc25s)                                for CycB~{p} =[Cdc25a]=> CycB.
MA(V2)                                     for CycB~{p} => _.
(MA(kw5p),MA(kw6))                         for _ <=> Wee1.
MA((kw2p+kw2s*[CycB])/(Jw2 + [Wee1]))      for Wee1 => Wee1~{p}.
MA(kw1/(Jw1+[Wee1~{p}]))                   for Wee1~{p} => Wee1.
MA(kwd)                                    for Wee1~{p} => _.
(kc3p+kc3s*[CycB])*(1 - [Cdc25a])/(Jc3 + 1 - [Cdc25a])
                                           for _ => Cdc25a.
MM(kc4,Jc4)                                for Cdc25a => _.


% Specification
check_ltl(oscil([CycA],4,2) & oscil([CycB],4,3.5)
& oscil([CycD],4,0.4) & oscil([CycE],4,1)).
```

9.2. Mammalian Circadian Clock

```
% mRNA
vsP*[Bmal1_nucl]^n/(KAP^n+[Bmal1_nucl]^n)
for _=[Bmal1_nucl]=>mPER.

vmP*[mPER]/(KmP+[mPER])+kdmp*[mPER]
for mPER=>_.

vsC*[Bmal1_nucl]^n/(KAC^n+[Bmal1_nucl]^n)
for _=[Bmal1_nucl]=>mCRY.
```

```
vmC*[mCRY]/(KmC+[mCRY])+kdmc*[mCRY]
for mCRY=>_.

vsB*KIB^m/(KIB^m+[REVERB_nucl]^m)
for _=>mBmal1.

vmB*[mBmal1]/(KmB+[mBmal1])+kdmb*[mBmal1]
for mBmal1=>_.

vsR*[Bmal1_nucl]^h/(KAR^h+[Bmal1_nucl]^h)
for _=[Bmal1_nucl]=>mREVERB.

vmR*[mREVERB]/(KmR+[mREVERB])+kdmr*[mREVERB]
for mREVERB=>_.

% Proteins
ksP*[mPER]
for _=[mPER]=>PER_cyto.

V2P*[PER_cyto~{p}]/(Kdp+[PER_cyto~{p}])
for PER_cyto~{p}=>PER_cyto.

ka4*[PER_cyto-CRY_cyto]
for PER_cyto-CRY_cyto=>PER_cyto+CRY_cyto.

V1P*[PER_cyto]/(Kp+[PER_cyto])
for PER_cyto=>PER_cyto~{p}.

kdn*[PER_cyto]
for PER_cyto=>_.

ka3*[PER_cyto]*[CRY_cyto]
for PER_cyto+CRY_cyto=>PER_cyto-CRY_cyto.

kdn*[PER_cyto~{p}]+vdPC*[PER_cyto~{p}]/(Kd+[PER_cyto~{p}])
for PER_cyto~{p}=>_.

ksC*[mCRY]
for _=[mCRY]=>CRY_cyto.

V2C*[CRY_cyto~{p}]/(Kdp+[CRY_cyto~{p}])
for CRY_cyto~{p}=>CRY_cyto.

V1C*[CRY_cyto]/(Kp+[CRY_cyto])
for CRY_cyto=>CRY_cyto~{p}.

kdnc*[CRY_cyto]
for CRY_cyto=>_.
```

```
vdCC*[CRY_cyto~{p}]/(Kd+[CRY_cyto~{p}])+kdn*[CRY_cyto~{p}]
for CRY_cyto~{p}=>_.

V2PC*[(PER_cyto-CRY_cyto)~{p}]/(Kdp+[(PER_cyto-CRY_cyto)~{p}])
for (PER_cyto-CRY_cyto)~{p}=>PER_cyto-CRY_cyto.

V1PC*[PER_cyto-CRY_cyto]/(Kp+[PER_cyto-CRY_cyto])
for PER_cyto-CRY_cyto=>(PER_cyto-CRY_cyto)~{p}.

ka2*[PER_nucl-CRY_nucl]
for PER_nucl-CRY_nucl=>PER_cyto-CRY_cyto.

ka1*[PER_cyto-CRY_cyto]
for PER_cyto-CRY_cyto=>PER_nucl-CRY_nucl.

kdn*[PER_cyto-CRY_cyto]
for PER_cyto-CRY_cyto=>_.

V4PC*[(PER_nucl-CRY_nucl)~{p}]/(Kdp+[(PER_nucl-CRY_nucl)~{p}])
for (PER_nucl-CRY_nucl)~{p}=>PER_nucl-CRY_nucl.

V3PC*[PER_nucl-CRY_nucl]/(Kp+[PER_nucl-CRY_nucl])
for PER_nucl-CRY_nucl=>(PER_nucl-CRY_nucl)~{p}.

ka8*[In]
for In=>Bmal1_nucl+PER_nucl-CRY_nucl.

ka7*[Bmal1_nucl]*[PER_nucl-CRY_nucl]
for Bmal1_nucl+PER_nucl-CRY_nucl=>In.

kdn*[PER_nucl-CRY_nucl]
for PER_nucl-CRY_nucl=>_.

vdPCC*[(PER_cyto-CRY_cyto)~{p}]/(Kd+[(PER_cyto-CRY_cyto)~{p}])+kdn*[(PER_cyto-CRY_cyto)~{p}]
for (PER_cyto-CRY_cyto)~{p}=>_.

vdPCN*[(PER_nucl-CRY_nucl)~{p}]/(Kd+[(PER_nucl-CRY_nucl)~{p}])+kdn*[(PER_nucl-CRY_nucl)~{p}]
for (PER_nucl-CRY_nucl)~{p}=>_.

ksB*[mBmal1]
for _=[mBmal1]=>Bmal1_cyto.

V2B*[Bmal1_cyto~{p}]/(Kdp+[Bmal1_cyto~{p}])
for Bmal1_cyto~{p}=>Bmal1_cyto.

V1B*[Bmal1_cyto]/(Kp+[Bmal1_cyto])
for Bmal1_cyto=>Bmal1_cyto~{p}.
```

```
ka6*[Bmal1_nucl]
for Bmal1_nucl=>Bmal1_cyto.

ka5*[Bmal1_cyto]
for Bmal1_cyto=>Bmal1_nucl.

kdn*[Bmal1_cyto]
for Bmal1_cyto=>_.

vdBC*[Bmal1_cyto~{p}]/(Kd+[Bmal1_cyto~{p}])+kdn*[Bmal1_cyto~{p}]
for Bmal1_cyto~{p}=>_.

V4B*[Bmal1_nucl~{p}]/(Kdp+[Bmal1_nucl~{p}])
for Bmal1_nucl~{p}=>Bmal1_nucl.

V3B*[Bmal1_nucl]/(Kp+[Bmal1_nucl])
for Bmal1_nucl=>Bmal1_nucl~{p}.

kdn*[Bmal1_nucl]
for Bmal1_nucl=>_.

vdBN*[Bmal1_nucl~{p}]/(Kd+[Bmal1_nucl~{p}])+kdn*[Bmal1_nucl~{p}]
for Bmal1_nucl~{p}=>_.

vdIN*[In]/(Kd+[In])+kdn*[In]
for In=>_.

ksR*[mREVERB]
for _=[mREVERB]=>REVERB_cyto.

ka10*[REVERB_nucl]
for _=[REVERB_nucl]=>REVERB_cyto.

(ka9+kdn)*[REVERB_cyto]+vdRC*[REVERB_cyto]/(Kd+[REVERB_cyto])
for REVERB_cyto=>_.

ka9*[REVERB_cyto]
for _=[REVERB_cyto]=> REVERB_nucl.

(ka10+kdn)*[REVERB_nucl]+vdRN*[REVERB_nucl]/(Kd+[REVERB_nucl])
for REVERB_nucl=>_.

% Light-dark entraining
macro(vsP,sq_wave(vsP_light,12,vsP_dark,12)).


% Specification
check_ltl(period(mPER,24) & period(mCRY,24) & period(mBmal1,24)
& period(mREVERB,24)).
```

9.3. P53/Mdm2 DNA-damage Repair System

```
% p53
(ks53,MA(kd53p)) for _ <=> p53.
MA(kf) for p53  =[Mdm2::n]=> p53~{u}.
MA(kr) for p53~{u}  => p53.
MA(kd53p) for p53~{u} => _.
MA(kf) for p53~{u}  =[Mdm2::n]=> p53~{uu}.
MA(kr) for p53~{uu}  => p53~{u}.
(kd53+kd53p)*[p53~{uu}] for p53~{uu} => _.

% DNA damage
(kDNA*IR,MM(kdDNA*p53tot,Jdna)) for _ <=> DNAdam.

add_event(Time>=10,IR,1).
add_event(Time>=20,IR,0).

% Mdm2
(ks2p,MA(kd2p)) for _ <=> Mdm2::c.
ks2*p53totˆmp/(Jsˆmp+p53totˆmp) for _ =[p53]=> Mdm2::c.

(kph*[Mdm2::c]/(Jph+p53tot),MA(kdeph)) for Mdm2::c <=> Mdm2~{p}::c.
MA(kd2p) for Mdm2~{p}::c => _.

(MA(ko),MA(ki)) for Mdm2::n <=> Mdm2~{p}::c.

kd2p_n*[Mdm2::n] for Mdm2::n => _.
[Mdm2::n]*[DNAdam]*kd2pp_n/(Jdam+[DNAdam]) for Mdm2::n =[DNAdam]=> _.


% Specification
check_ltl(G([DNAdam]=0) -> G(d([p53])/dt = 0 & d([Mdm2::n])/dt = 0) &
G(([DNAdam]>0.2) -> F(oscil([p53],1) & F(oscil([Mdm2],1)))) &
G(oscil([p53],1)-> X((! oscil([p53],1)) U (oscil([Mdm2::n],1))))).
```

9.4. Irinotecan Metabolism

```
injection for _ => CPT11.

k1*[CPT11]*CES/(Km1+[CPT11]) for CPT11 => SN38.

k2*UGT1A1*[SN38]ˆnir/(Km2ˆnir+[SN38]ˆnir) for SN38 => SN38G.

kpgp2*[ABCG2]*[CPT11]/([CPT11]+Kpgp2) for CPT11 =[ABCG2]=> _.

kpgp1*[ABCG2]*[SN38]/([SN38]+Kpgp1) for SN38 =[ABCG2]=> _.
```

```
(MA(kcompl)*DNAfree,MA(kdecompl)) for SN38 + TOP1 <=> TOP1cc.

MA(kdam) for TOP1cc => DNAdam.

MA(kd3) for SN38G => _.

MA(kdtop1) for TOP1 => _.

top1 for _ => TOP1.

delete_rules(_ => DNAdam).
delete_event(Time>=10,IR,1).
delete_event(Time>=20,IR,0).


% Specification
check_ltl(G([CPT11]=0) -> G([DNAdam] = 0) &
G([CPT11]>10) -> FG([DNAdam] > 3.5)).
```

9.5. Irinotecan Injection Control

```
add_event(Time>=1,injection,0).
add_event(Time>=interval,injection,10).
add_event(Time>=interval+1,injection,0).
add_event(Time>=2*interval,injection,10).
add_event(Time>=2*interval+1,injection,0).
add_event(Time>=3*interval,injection,10).
add_event(Time>=3*interval+1,injection,0).
add_event(Time>=4*interval,injection,10).
add_event(Time>=4*interval+1,injection,0).
```

Chapter 5

Spiking Neural Networks Modelled as Timed Automata

Elisabetta De Maria, Cinzia Di Giusto, and Laetitia Laversa
Journal of Natural Computing, 2019

Noname manuscript No.
(will be inserted by the editor)

Spiking Neural Networks modelled as Timed Automata

with parameter learning

Elisabetta De Maria · Cinzia Di Giusto · Laetitia Laversa

Received: date / Accepted: date

Abstract In this paper we address the issue of automatically learning parameters of spiking neural networks. Biological neurons are formalized as timed automata and synaptical connections are represented as shared channels among these automata. Such a formalism allows us to take into account several time-related aspects, such as the influence of past inputs in the computation of the potential value of each neuron, or the presence of the refractory period, a lapse of time immediately following the spike emission in which the neuron cannot emit. The proposed model is then formally validated: more precisely, we ensure that some relevant properties expressed as temporal logical formulae hold in the model. Once the validation step is accomplished, we take advantage of the proposed model to write an algorithm for learning synaptical weight values such that an expected behavior can be displayed. The technique we present takes inspiration from supervised learning ones: we compare the effective output of the network to the expected one and backpropagate proper corrective actions in the network. We develop several case studies including a mutual inhibition network.

Keywords Neural Networks · Parameter Learning · Timed Automata, Temporal Logic · Model Checking.

1 Introduction

The brain behaviour is the object of thorough studies: researchers are interested not only in the inner functioning of neurons (which are its elementary components), their interactions and the way these aspects participate to the ability to move, learn or remember, typical of living beings; but also in reproducing such capabilities (emulating nature), e.g., within robot controllers, speech/text/face recognition applications, etc. In order to achieve a detailed understanding of the brain functioning, both neurons behaviour and their interactions must be studied. Several models of the neuron behaviour have been proposed: some of them make neurons behave as binary threshold gates, other ones exploit a sigmoidal

Université Côte d'Azur, CNRS, I3S, France

transfer function, while, in many cases, differential equations are employed. According to [30,27], three different and progressive *generations* of neural networks can be recognised: (i) *first generation* models handle discrete inputs and outputs and their computational units are threshold-based transfer functions; they include McCulloch and Pitt's threshold gate model [29], the perceptron model[16], Hopfield networks [22], and Boltzmann machines [1]; (ii) *second generation* models exploit real valued activation functions, e.g., the sigmoid function, accepting and producing real values: a well known example is the multi-layer perceptron [9,33]; (iii) *third generation* networks are known as spiking neural networks. They extend second generation models treating time-dependent and real valued signals often composed by *spike trains*. Neurons may fire output spikes according to threshold-based rules which take into account input spike magnitudes and occurrence times [30].

The core of our analysis are *spiking neural networks* [17]. Because of the introduction of timing aspects they are considered closer to the actual brain functioning than other generations models. Spiking neurons emit spikes taking into account input impulses strength and their occurrence instants. Models of this sort are of great interest, not only because they are closer to natural neural networks behaviour, but also because the temporal dimension allows to represent information according to various *coding schemes* [31,30]: e.g., the amount of spikes occurred within a given time window (*rate* coding), the reception/absence of spikes over different synapses (*binary* coding), the relative order of spikes occurrences (*rate rank* coding), or the precise time difference between any two successive spikes (*timing* coding).

Several spiking neuron models have been proposed in the literature, having different complexities and capabilities. In [25], Izhikevich classifies spiking neuron models according to some *behaviour* (i.e., typical responses to an input pattern) that they should exhibit in order to be considered biologically relevant. The leaky integrate & fire (LI&F) model [26], where past inputs relevance exponentially decays with time, is one of the most studied neuron models because it is straightforward and easy to use [25,30]. On the other end of the spectrum, the Hodgkin-Huxley (H-H) model [21] is one of the most complex being composed by four differential equations comparing neurons to electrical circuits. In [25], the H-H model can reproduce all behaviours under consideration, but the simulation process is really expensive even for just a few neurons being simulated for a small amount of time. Our aim is to produce a neuron model being meaningful from a biological point of view but also amenable to formal analysis and verification, that could be therefore used to detect non-active portions within some network (i.e., the subset of neurons not contributing to the network outcome), to test whether a particular output sequence can be produced or not, to prove that a network may never be able to emit, to assess if a change to the network structure can alter its behaviour, or to investigate (new) learning algorithms which take time into account.

In this paper we focus on the *leaky integrate & fire* (LI&F) model originally proposed in [26]. It is a computationally efficient approximation of single-compartment model [25] and is abstracted enough to be able to apply formal verification techniques such as model-checking. Here we work on an extended version of the discretised formulation proposed in [14], which relies on the notion of logical time. Time is considered as a sequence of logical discrete instants, and an instant is a point in

time where external input events can be observed, computations can be done, and outputs can be emitted. The variant we introduce here takes into account some new time-related aspects, such as a lapse of time in which the neuron is not active, i.e., it cannot receive and emit. We encode LI&F networks into timed automata: we show how to define the behaviour of a single neuron and how to build a network of neurons. Timed automata [2] are finite state automata extended with timed behaviours: constraints are allowed to limit the amount of time an automaton can remain within a particular state, or the time interval during which a particular transition may be enabled. Timed automata networks are sets of automata that can synchronise over *channel* communications.

Our modelling of spiking neural networks consists of timed automata networks where each neuron is an automaton. Its behaviour consists in accumulating the weighted sum of inputs, provided by a number of ingoing weighted synapses, for a given amount of time. Then, if the *potential* accumulated during the last and previous accumulation periods overcomes a given threshold, the neuron fires an output over the outgoing synapse. Synapses are channels shared between the timed automata representing neurons, while *spike* emissions are represented by *broadcast synchronisations* occurring over such channels. Timed automata are also exploited to produce or *recognise* precisely defined spike sequences.

As a first main contribution, we analyse some intrinsic properties of the proposed model, e.g., the maximum threshold value allowing a neuron to emit, or the lack of inter-spike memory, preventing the behaviour of a neuron from being influenced by what happened before the last spike. Furthermore, we encode in temporal logics all the behaviours (or capabilities) a LI&F model should be able to reproduce according to Izhikevich and we exploit model checking to prove these behaviours are reproducible in our model. Izhikevich also identifies a set of behaviours which are not expected to be reproducible by any LI&F model. We prove these limits to hold for our model, too, and we provide, for each non-reproducible behaviour, an extension of the model allowing to reproduce it.

As a second main contribution, we exploit our automata-based modelling to propose a new methodology for parameter inference in spiking neural networks. In particular, our approach allows to find an assignment for the synaptical weights of a given neural network such that it can reproduce a given behaviour. We apply the proposed approach to find suitable parameters in mutual inhibition networks, a well studied class of networks in which the constituent neurons inhibit each other neuron's activity [28].

In this work we do not intend to treat classical classification problems of artificial intelligence, but we to focus on biological neural networks and aim at studying the behavior of small circuits. The basic biological hypothesis is that neurons in our brain tend to form some mini-circuits with a relevant structure and behavior, which are often referred as archetypes [14]. These small circuits (e.g., simple series, parallel composition, negative loop, inhibition of a behavior, etc.) can be seen as the constituting bricks of bigger networks. We are interested in finding parameters such that these small graphs behave as expected (e.g., for suitable parameters a negative loop exhibits an oscillating trend), not to compete with (simulation) approaches dealing with big networks.

This paper is an extended and revised version of the conference papers [11] and [12]. In particular, in section 5 we propose a refined version of the Advice Back-Propagation algorithm. Furthermore, we add a second learning technique

that is based on simulation instead of model checking. We also show that the two techniques can be combined.

The rest of the paper is organised as follows: in Section 2 we recall definitions of timed automata networks, temporal logics, and model checking; in Section 3 we describe our reference model, the LI&F one, and its encoding into timed automata networks; in Section 4 we study some intrinsic properties of the obtained model and we validate it against its ability of reproducing or not some behaviours; in Section 5 we develop the novel parameter learning approach and we introduce several case studies; in Section 6 we give an overview of the related work. Finally, Section 7 summarises our contribution and presents some future research directions.

2 Preliminaries

In this section we introduce the formal tools we adopt in the rest of the paper, namely timed automata and temporal logics.

2.1 Timed Automata.

Timed automata [2] are a powerful theoretical formalism for modelling and verifying real time systems. A timed automaton is an annotated directed (and connected) graph, with an initial node and provided with a finite set of non-negative real variables called *clocks*. Nodes (called *locations*) are annotated with *invariants* (predicates allowing to enter or stay in a location), arcs with *guards*, *communication labels*, and possibly with some variables upgrades and clock *resets*. Guards are conjunctions of elementary predicates of the form $x\ op\ c$, where $op \in \{>, \geq, =, <, \leq\}$, x is a clock, and c a (possibly parameterised) positive integer constant. As usual, the empty conjunction is interpreted as true. The set of all guards and invariant predicates will be denoted by G.

Definition 1 A *timed automaton* TA is a tuple $(L, l^0, X, \Sigma, Arcs, Inv)$, where

- L is a set of locations with $l^0 \in L$ the initial one
- X is the set of clocks,
- Σ is a set of communication labels,
- $Arcs \subseteq L \times (G \cup \Sigma \cup U) \times L$ is a set of arcs between locations with a guard in G, a communication label in $\Sigma \cup \{\varepsilon\}$, and a set of variable upgrades (e.g., clock resets);
- $Inv : L \to G$ assigns invariants to locations.

It is possible to define a synchronised product of a set of timed automata that work and synchronise in parallel. The automata are required to have disjoint sets of locations, but may share clocks and communication labels which are used for synchronisation. We restrict communications to be *broadcast* through labels $b!, b? \in \Sigma$, meaning that a set of automata can synchronise if one is emitting; notice that a process can always emit (e.g., $b!$) and the receivers ($b?$) must synchronise if they can.

Locations can be normal, urgent or committed. Urgent locations force the time to freeze, committed ones freeze time and the automaton must leave the location as soon as possible, i.e., they have higher priority.

The synchronous product $TA_1 \parallel \ldots \parallel TA_n$ of timed automata, where $TA_j = (L_j, l_j^0, X_j, \Sigma_j, Arcs_j, Inv_j)$ and L_j are pairwise disjoint sets of locations for each $j \in [1, \ldots, n]$, is the timed automaton

$$TA = (L, l^0, X, \Sigma, Arcs, Inv)$$

such that:

- $L = L_1 \times \ldots \times L_n$ and $l^0 = (l_1^0, \ldots, l_n^0)$, $X = \bigcup_{j=1}^{n} X_j$, $\Sigma = \bigcup_{j=1}^{n} \Sigma_j$,
- $\forall l = (l_1, \ldots, l_n) \in L$: $Inv(l) = \bigwedge_j Inv_j(l_j)$,
- $Arcs$ is the set of arcs $(l_1, \ldots, l_n) \xrightarrow{g,a,r} (l'_1, \ldots, l'_n)$ such that for all $1 \leq j \leq n$ then $l'_j = l_j$.

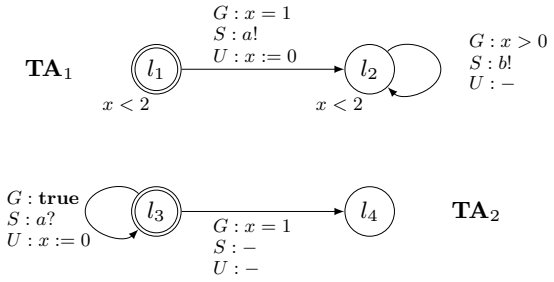
Its semantics is the one of the underlying timed automaton TA with the following notations. A location is a vector $l = (l_1, \ldots, l_n)$. We write $l[l'_j/l_j, j \in S]$ to denote the location l in which the j^{th} element l_j is replaced by l'_j, for all j in some set S. A valuation is a function ν from the set of clocks to the non-negative reals. Let $\mathbb{V}$ be the set of all clock valuations, and $\nu_0(x) = 0$ for all $x \in X$. We shall denote by $\nu \vDash F$ the fact that the valuation ν satisfies (makes true) the formula F. If r is a clock reset, we shall denote by $\nu[r]$ the valuation obtained after applying the clock reset $r \subseteq X$ to ν; and if $d \in \mathbb{R}_{>0}$ is a delay, $\nu + d$ is the valuation such that, for any clock $x \in X$, $(\nu + d)(x) = \nu(x) + d$.

The semantics of a synchronous product $TA_1 \parallel \ldots \parallel TA_n$ is defined as a timed transition system $(S, s_0, \rightarrow)$, where $S = (L_1 \times, \ldots \times L_n) \times \mathbb{V}$ is the set of states, $s_0 = (l^0, \nu_0)$ is the initial state, and $\rightarrow \subseteq S \times S$ is the transition relation defined by:

- (silent): $(l, \nu) \rightarrow (l', \nu')$ if there exists $l_i \xrightarrow{g,\varepsilon,r} l'_i$, for some i, such that $l' = l[l'_i/l_i]$, $\nu \vDash g$ and $\nu' = \nu[r]$,
- (broadcast): $(\bar{l}, \nu) \rightarrow (\bar{l'}, \nu')$ if there exists an output arc $l_j \xrightarrow{g_j,b!,r_j} l'_j \in Arcs_j$ and a (possibly empty) set of input arcs of the form $l_k \xrightarrow{g_k,b?,r_k} l'_k \in Arcs_k$ such that for all $k \in K = \{k_1, \ldots, k_m\} \subseteq \{l_1, \ldots, l_n\} \backslash \{l_j\}$, the size of K is maximal, $\nu \vDash \bigwedge_{k \in K \cup \{j\}} g_k$, $l' = l[l'_k/l_k, k \in K \cup \{j\}]$ and $\nu' = \nu[r_k, k \in K \cup \{j\}]$;
- (timed): $(l, \nu) \rightarrow (l, \nu + d)$ if $\nu + d \vDash Inv(l)$.

The valuation function ν is extended to handle a set of shared bounded integer variables: predicates concerning such variables can be part of edges guards or locations invariants, moreover variables can be updated on edges firings but they cannot be assigned to or from clocks.

Example 1 In Figure 1 we consider the network of timed automata TA_1 and TA_2 with broadcast communications, and we give a possible run. TA_1 and TA_2 start in the l_1 and l_3 locations, respectively, so the initial state is $[(l_1, l_3);\ x = 0]$. A *timed* transition produces a delay of 1 time unit, making the system move to state $[(l_1, l_3);\ x = 1]$. A *broadcast* transition is now enabled, making the system move to state $[(l_2, l_3);\ x = 0]$, broadcasting over channel a and resetting the x clock. Two successive *timed* transitions (0.5 time units) followed by a *broadcast* one will eventually lead the system to state $[(l_2, l_4);\ x = 1]$. ◇



(a) The timed automata network $TA_1 \parallel TA_2$.

$[(l_1, l_3);\ x = 0]$
↓
$[(l_1, l_3);\ x = 1]$
↓
$[(l_2, l_3);\ x = 0]$
↓
$[(l_2, l_3);\ x = 0.5]$
↓
$[(l_2, l_3);\ x = 1]$
↓
$[(l_2, l_4);\ x = 1]$

(b) A possible run.

Fig. 1: A network of timed automata with a possible run.

Throughout our modelling, we have used the specification and analysis tool `Uppaal` [4], which provides the possibility of designing and simulating timed automata networks on top of the ability of testing networks against temporal logic formulae. All figures depicting timed automata follow the graphic conventions of the tool (e.g., initial states are denoted with a double circle).

2.2 Temporal Logics and Model Checking

Model checking is one of the most common approaches to the verification of software and hardware (distributed) systems [8]. It allows to automatically prove whether a system verifies or not a given specification. In order to apply such a technique, the system at issue should be encoded as a finite transition system and the specification should be written using propositional temporal logic. Formally, a transition system over a set AP of atomic propositions is a tuple $M = (Q, T, L)$, where Q is a finite set of states, $T \subseteq Q \times Q$ is a total transition relation, and $L : Q \to 2^{AP}$ is a labelling function that maps every state into the set of atomic propositions that hold at that state.

Temporal formulae describe the dynamical evolution of a given system. The computation tree logic CTL* allows to describe properties of computation trees. Its formulas are obtained by (repeatedly) applying boolean connectives ($\wedge$, $\vee$, $\neg$, $\rightarrow$), *path quantifiers*, and *state quantifiers* to atomic formulas. The path quantifier **A** (resp., **E**) can be used to state that all the paths (resp., some path) starting from a given state have some property. The state quantifiers are **X** (next time), which specifies that a property holds at the next state of a path, **F** (sometimes in the future), which requires a property to hold at some state on the path, **G** (always in the future), which imposes that a property is true at every state on the path, and **U** (until), which holds if there is a state on the path where the second of its argument properties holds and, at every preceding state on the path, the first of its two argument properties holds. Given two formulas φ_1 and φ_2, in the

rest of the paper we use the shortcut $\varphi_1 \rightsquigarrow \varphi_2$ to denote the liveness property $AG(\varphi_1 \rightarrow AF\varphi_2)$, which can be read as "$\varphi_1$ always leads to φ_2 ".

The branching time logic CTL is a fragment of CTL* that allows quantification over the paths starting from a given state. Unlike CTL*, it constrains every state quantifier to be immediately preceded by a path quantifier.

Given a transition system $M = (Q, T, L)$, a state $q \in Q$, and a temporal logic formula φ expressing some desirable property of the system, the *model checking problem* consists of establishing whether φ holds at q or not, namely, whether $M, q \models \varphi$.

3 Leaky Integrate and Fire Model and Mapping to Timed Automata

Spiking neural networks [27] are modelled as directed weighted graphs where vertices are computational units and edges represent *synapses*. The signals propagating over synapses are trains of impulses: *spikes*. Synapses may modulate these signals according to their weight: *excitatory* if positive, or *inhibitory* if negative.

The dynamics of neurons is governed by their *membrane potential* (or, simply, *potential*), representing the difference of electrical potential across the cell membrane. The membrane potential of each neuron depends on the spikes received over the ingoing synapses. Both current and past spikes are taken into account, even if old spikes contribution is lower. In particular, the *leak factor* is a measure of the neuron memory about past spikes. The neuron outcome is controlled by the algebraic difference between its membrane potential and its *firing threshold*: it is enabled to fire (i.e., emit an output impulse over *all* outgoing synapses) only if such a difference is non-negative. Spike propagation is assumed to be instantaneous. Immediately after each emission the neuron membrane potential is reset and the neuron stays in a *refractory period* for a given amount of time. During this period it has no dynamics: it cannot increase its potential as any received spike is lost and therefore it cannot emit any spike.

Definition 2 (Spiking Integrate and Fire Neural Network) *A spiking integrate and fire neural network* is a tuple (V, A, w), where:

- V are spiking integrate and fire neurons,
- $A \subseteq V \times V$ are synapses,
- $w : A \rightarrow \mathbb{Q} \cap [-1, 1]$ is the synapse weight function associating to each synapse (u, v) a weight $w_{u,v}$.

We distinguish three disjoint sets of neurons: V_i (input neurons), V_{int} (intermediary neurons), and V_o (output neurons), with $V = V_i \cup V_{int} \cup V_o$.
A spiking integrate and fire neuron v is characterized by a parameter tuple

$$(\theta_v, \tau_v, \lambda_v, p_v, y_v),$$

where:

- $\theta_v \in \mathbb{N}$ is the *firing threshold*,
- $\tau_v \in \mathbb{N}^+$ is the *refractory* period,
- $\lambda_v \in \mathbb{Q} \cap [0, 1]$ is the *leak factor*.

The dynamics of a *spiking integrate and fire neuron* v is given by:

– $p_v : \mathbb{N} \to \mathbb{Q}_0^+$ is the [membrane] *potential* function defined as

$$p_v(t) = \begin{cases} \sum_{i=1}^{m} w_i \cdot x_i(t), & \text{if } p_v(t-1) \geqslant \theta_v \\ \sum_{i=1}^{m} w_i \cdot x_i(t) + \lambda_v \cdot p_v(t-1), & \text{otherwise.} \end{cases}$$

with $p_v(0) = 0$ and where $x_i(t) \in \{0, 1\}$ is the signal received at the time t by the neuron through its i^{th} out of m input synapses (observe that the past potential is multiplied by the leak factor while current inputs are not weakened),

– $y_v : \mathbb{N} \to \{0, 1\}$ is the neuron output function, defined as

$$y_v(t) = \begin{cases} 1 & \text{if } p_v(t) \geqslant \theta_v \\ 0 & \text{otherwise.} \end{cases}$$

As shown in the previous definition, the set of neurons of a spiking integrate and fire neural network can be classified into input, intermediary, and output ones. Each input neuron can only receive as input external signals (and not other neurons' output). The output of each output neuron is considered as an output for the network. Output neurons are the only ones whose output is not connected to other neurons.

We present here our modelling of spiking integrate and fire neural networks (in the following denoted as neural networks) via timed automata networks. Let $S = (V, A, w)$ be a neural network, G be a set of *input generator* neurons (these fictitious neurons are connected to input neurons and generate input sequences for the network), and O be a set of *output consumer* neurons (these fictitious neurons are connected to the broadcast channel of each output neuron and aim at consuming their emitted spikes). The corresponding timed automata network is obtained as the synchronous product of the encoding of input generator neurons, the neurons of the network (referred as standard neurons in the following), and output consumers neurons. More formally:

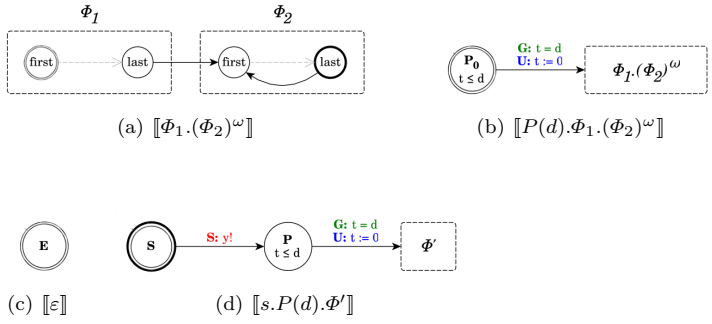
$$[\![S]\!] = \left(\Big\|_{n_g \in G} [\![n_g]\!] \right) \| \left(\Big\|_{v_j \in V} [\![v_j]\!] \right) \| \left(\Big\|_{n_c \in O} [\![n_c]\!] \right)$$

Input generators. The behaviour of input generator neurons is part of the specification of the network. Here we define two kinds of input behaviours: regular and non-deterministic ones. For each family, we provide an encoding into timed automata.

Regular input sequences. Spike trains are "regular" sequences of spikes and pauses: spikes are instantaneous while pauses have a non-null duration. Sequences can be *empty*, *finite* or *infinite*. After each spike there must be a pause, except when the spike is the last event of a finite sequence. Infinite sequences are composed by two parts: a finite and arbitrary prefix and an infinite and periodic part composed by a finite sequence of *spike–pause* pairs which is repeated infinitely often. More formally, such sequences are given in terms of the following grammar:

$$\begin{aligned} G &::= \Phi.(\Phi)^\omega \mid P(d).\Phi.(\Phi)^\omega \\ \Phi &::= s.P(d).\Phi \mid \varepsilon \end{aligned}$$

with s representing a spike and $P(d)$ a pause of duration d. It is possible to generate an emitter automaton for any regular input sequence:



(a) $[\![\Phi_1.(\Phi_2)^\omega]\!]$ (b) $[\![P(d).\Phi_1.(\Phi_2)^\omega]\!]$

(c) $[\![\varepsilon]\!]$ (d) $[\![s.P(d).\Phi']\!]$

Fig. 2: Representation of the encoding of an input sequence

Definition 3 (Input generator) Let $I \in \mathcal{L}(G)$ be a word over the language generated by IS, then its encoding into timed automata is $[\![I]\!] = (L(I),\ first(I),\ \{t\},\ \{y\},\ Arcs(I),\ Inv(I))$. It is inductively defined as follows:

- if $I := \Phi_1.(\Phi_2)^\omega$
  - $L(I) = L(\Phi_1) \cup L(\Phi_2)$, where $last(\Phi_2)$ is *urgent*
  - $first(I) = first(\Phi_1)$
  - $Arcs(I) = Arcs(\Phi_1) \cup Arcs(\Phi_2) \cup$
    $\{(last(\Phi_1), \mathbf{true}, \varepsilon, \emptyset, first(\Phi_2)),$
    $(last(\Phi_1), \mathbf{true}, \varepsilon, \emptyset, first(\Phi_2))\}$
  - $Inv(I) = Inv(\Phi_1) \cup Inv(\Phi_2)$
- if $I := P(d).\Phi_1.(\Phi_2)^\omega$
  - $L(I) = \{\mathbf{P}_0\} \cup L(\Phi_1) \cup L(\Phi_2)$, where $last(\Phi_2)$ is *urgent*
  - $first(I) = \mathbf{P}_0$
  - $Arcs(I) = Arcs(\Phi_1) \cup Arcs(\Phi_2) \cup$
    $\{(\mathbf{P}_0, t \leq d,\ , \{t := 0\}, first(\Phi_1)),$
    $(last(\Phi_1), \mathbf{true}, \varepsilon, \emptyset, first(\Phi_2)),$
    $(last(\Phi_1), \mathbf{true}, \varepsilon, \emptyset, first(\Phi_2))\}$
  - $Inv(I) = \{\mathbf{P}_0 \mapsto t \leq d\} \cup Inv(\Phi_1) \cup Inv(\Phi_2)$
- if $\Phi := \varepsilon$
  - $L(\Phi) = \{\mathbf{E}\}$
  - $first(\Phi) = last(\Phi) = \mathbf{E}$
  - $Arcs(\Phi) = \emptyset$
  - $Inv(\Phi) = \emptyset$
- if $\Phi := s.P(d).\Phi'$
  - $L(\Phi) = \{\mathbf{S}, \mathbf{P}\} \cup L(\Phi')$
  - $first(\Phi) = \mathbf{S}, \quad last(\Phi) = last(\Phi')$
  - $Arcs(\Phi) = Arcs(\Phi') \cup \{(\mathbf{S}, \mathbf{true}, y!, \emptyset, \mathbf{P}),$
    $(\mathbf{P}, t = d, \varepsilon, \{t := 0\}, first(\Phi'))\}$
  - $Inv(\Phi) = \{\mathbf{P} \mapsto t \leq d\} \cup Inv(\Phi')$

Figure 2 depicts the shape of input generators. Figure 2(a) shows the generator $[\![I]\!]$, obtained from $I := \Phi_1.(\Phi_2)^\omega$. The edge connecting the last state of $[\![\Phi_2]\!]$ to the first one allows Φ_2 to be repeated infinitely often. Figure 2(b) shows the case of an input sequence $I := P(d).\Phi_1.(\Phi_2)^\omega$ beginning with a pause $P(d)$: in this

case, the initial location of $[\![I]\!]$ is $\mathbf{P}_0$, which imposes a delay of d time units. The remainder of the input sequence is encoded as for the previous case. Figure 2(c) shows the induction basis for encoding a sequence Φ, i.e., the case $\Phi := \varepsilon$. It is encoded as a location **E** having no edge. Finally, Figure 2(d) shows the case of a non-empty spike–pause pair sequence $\Phi := s.P(d).\Phi'$. It consists of an *urgent* location **S**: when the automaton moves from **S**, a spike is fired over channel y and the automaton moves to location **P**, representing a silent period. After that, the automaton proceeds with the encoding of Φ'.

Non-deterministic input sequences. This kind of input sequences is useful when no assumption is available on neuron inputs. These are random sequences of spikes separated by at least T_{min} time units.

Such sequences can be generated by an automaton defined as follows:

Definition 4 (Non-deterministic input generator) A non-deterministic input generator I_{nd} is a tuple

$$(L, \mathbf{B}, X, \Sigma, Arcs, Inv),$$

with:

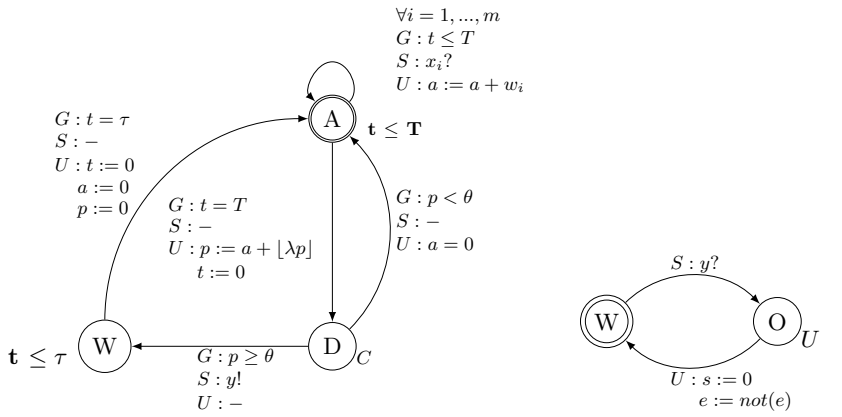
- $L = \{\mathbf{B}, \mathbf{S}, \mathbf{W}\}$, with **S** urgent,
- $X = \{t\}$
- $\Sigma = x$
- $Arcs = \{(\mathbf{B}, t = D, x!, \emptyset, \mathbf{S}), (\mathbf{S}, \mathbf{true}, \varepsilon, \{t := 0\}, \mathbf{W}), (\mathbf{W}, t > T_{min}, x!, \emptyset, \mathbf{S})\}$
- $Inv(B) = (t \leq D)$

where D is the initial delay.

The behavior of such a generator depends on clock t and broadcast channel x, and can be summarized as follows: it waits in location **B** an arbitrary amount of time before moving to location **S**, firing its first spike over channel x. Since location **S** is *urgent*, the automaton instantaneously moves to location **W**, resetting clock t. Finally, from location **W**, after an arbitrary amount of time t, it moves to location **S**, firing a spike. Notice that an initial delay D may be introduced by adding the invariant $t \leq D$ to the location **B** and the guard $t = D$ on the edge $(\mathbf{B} \rightarrow \mathbf{S})$.

Standard neurons. The neuron is a computational unit behaving as follows: i) it accumulates potential whenever it receives input spikes within a given *accumulation period*, ii) if the accumulated potential is greater than the *threshold*, it emits an output spike, iii) it waits during a *refractory period*, and restarts from i). Observe that the accumulation period is not present in the definition of neuron (Definition 2). It is indeed introduced here to slice time and therefore discretise the decrease of the potential value due to the leak factor. We assume that two input spikes on the same synapse cannot be received within the same accumulation period (i.e., the accumulation period is shorter than the minimum refractory period of the input neurons of the network). Next, we give the encoding of neurons into timed automata.

Definition 5 Given a neuron $v = (\theta, \tau, \lambda, p, y)$ with m input synapses, its encoding into timed automata is $\mathcal{N} = (L, \mathbf{A}, X, Var, \Sigma, Arcs, Inv)$ with:



(a) Neuron model.

(b) Output consumer automaton.

Fig. 3: Automata for standard neuron and output consumer.

- $L = \{\mathbf{A}, \mathbf{W}, \mathbf{D}\}$ with $\mathbf{D}$ committed,
- $X = \{t\}$
- $Var = \{p, a\}$
- $\Sigma = \{x_i \mid i \in [1..m]\} \cup \{y\}$,
- $Arcs = \{(\mathbf{A}, t \leq T, x_i?, \{a := a + w_i\}, \mathbf{A}) \mid i \in [1..m]\} \cup \{(\mathbf{A}, t = T, \ , \{p := a + \lfloor \lambda p \rfloor\}, \mathbf{D}),$
  $(\mathbf{D}, p < \theta, \ , \{a := 0\}, \mathbf{A}), (\mathbf{D}, p \geq \theta, y!, \ , \mathbf{W}),$
  $(\mathbf{W}, t = \tau, \ , \{a := 0, t := 0, p := 0\}, \mathbf{A})\}$;
- $Inv(\mathbf{A}) = t \leq T, Inv(\mathbf{W}) = t \leq \tau, Inv(\mathbf{D}) = \mathbf{true}$.

The neuron behavior, described by the automaton in Figure 3(a), depends on the following channels, variables and clocks:

- x_i for $i \in [1..m]$ are the m input channels,
- y is the broadcast channel used to emit the output spike,
- $p \in \mathbb{N}$ is the current potential value, initially set to zero,
- $a \in \mathbb{N}$ is the weighted sum of input spikes occurred within the *current* accumulation period; it equals zero at the beginning of each round.

The behaviour of the automaton modelling neuron v can be summed up as follows:

- the neuron keeps waiting in state $\mathbf{A}$ (for Accumulation) for input spikes while $t \leqslant T$ and, whenever it receives a spike on input x_i, it updates a with $a := a + w_i$;
- when $t = T$, the neuron moves to state $\mathbf{D}$ (for Decision), resetting t and updating p according to the potential function given in Definition 2:

$$p := a + \lfloor \lambda \cdot p \rfloor$$

  Since state $\mathbf{D}$ is *committed*, it does not allow time to progress, so, from this state, the neuron can move back to state $\mathbf{A}$ resetting a if the potential has not

reached the threshold $p < \theta$, or it can move to state **W**, firing an output spike, otherwise;
- the neuron remains in state **W** (for Wait) for τ time units (τ is the length of the refractory period) and then it moves back to state **A** resetting a, p and t.

Output consumers. In order to have a complete modelling of a spiking neural network, for each output neuron we build an *output consumer* automaton O_y. The automaton, whose formal definition is straightforward, is shown in Figure 3(b). The consumer waits in location **W** for the corresponding output spikes on channel y and, as soon as it receives the spike, it moves to location **O**. This location is only needed to simplify model checking queries. Since it is urgent, the consumer instantly moves back to location **W** resetting s, the clock measuring the elapsed time since last emission, and setting e to its negation, with e being a *boolean variable* which differentiates each emission from its successor.

Definition 6 (Output consumer) An output consumer is a timed automaton

$$\mathcal{N} = (L, \mathbf{W}, X, Var, \Sigma, Arcs, Inv)$$

with:

- $L = \{\mathbf{W}, \mathbf{O}\}$ with O urgent,
- $X = \{s\}$
- $Var = \{e\}$
- $\Sigma = \{y_i \mid y_i \text{ is an output neuron}\}$
- $Arcs = \{(\mathbf{W}, , y?, , \mathbf{O}),$
  $(\mathbf{O}, s := 0, , \{e := not(e)\}, \mathbf{W})\}$
- $Inv(\mathbf{W}) = \mathbf{true}, Inv(\mathbf{O}) = \mathbf{true}$.

We have a complete implementation of the spiking neural network model proposed in the paper via the tool `Uppaal`. It can be found on the web page [6]. We have validated our neuron model against some characteristic properties studied in [25] (tonic spiking, excitability, integrator, etc.). These properties have been formalised in temporal logics and checked via model-checking tools.

Observe that, since we rely on a discrete time, we could have used tick automata [19], a variant of Büchi automata where a special clock models the discrete flow of time. However, to the best of our knowledge, no existing tool allows to implement such automata. We decided to opt for timed automata in order to have an effective implementation of our networks to be exploited in parameter learning algorithms.

4 Validation of the model

In this section we show some properties of the neuron model of Definition 5. The first group of properties are structural. We can compute a minimum value such that any neuron, having a threshold greater than or equal to it, will never be able to fire.

Property 1 Let $\mathcal{N} = (\theta, \tau, \lambda, p, y)$ be a neuron and a_{max} be the maximum value received during each accumulation period. Then, if $\theta \geq \frac{a_{max}}{1-\lambda}$, the neuron is not able to fire.

Proof Without loss of generality, we suppose that, during each accumulation period, $\mathcal{N}$ receives the maximum possible input a_{max}. Then, its potential function is:

$$p_n = a_{max} + \lfloor \lambda \cdot p_{n-1} \rfloor$$

which is always lower than or equal to its undiscretized version:

$$p_n \leq {p'}_n = a_{max} + \lambda \cdot p'_{n-1}$$

The same inequality can be written in explicit form:

$$p_n \leq {p'}_n = \sum_{k=0}^{n} a_{n-k} \cdot \lambda^k$$

and, since we assumed the neuron always receives a_{max}, a_{n-k} is constant and does not depend on k:

$$p_n \leq a_{max} \cdot \sum_{k=0}^{n} \lambda^k$$

The rightmost factor is a geometric series:

$$p_n \leq a_{max} \cdot \frac{1 - \lambda^n}{1 - \lambda}$$

which reaches its maximum value $\frac{1}{1-\lambda}$ for $n \to \infty$, therefore:

$$p_n \leq \frac{a_{max}}{1 - \lambda}.$$

Thus, if $\theta \geq \frac{a_{max}}{1-\lambda}$, it is impossible for the neuron potential to reach the threshold and, consequently, the neuron cannot fire. □

In what follows, we only consider neurons that respect the previous constraint.

Apart from the minimum threshold, we can also quantify the amount of time that the neuron requires to complete an accumulate–fire–rest cycle. We show that there exists a minimum delay between neuron emissions.

Property 2 Let $\mathcal{N} = (\theta, \tau, \lambda, p, y)$ be a neuron. Then the time difference between successive firings cannot be lower than $T + \tau$.

Proof Let $A_n = \sum_{k=1}^{T} a_{k+t_0}$ be the sum of weighted inputs during the n-th *accumulation period*, then the neuron behaviour can be described as follows:

$$p_n = A_n + \lfloor \lambda \cdot p_{n-1} \rfloor$$

which is the potential value after the n-th accumulation period. If the neuron eventually fires an output spike, then there exists $\hat{n} > 0$ such that:

$$\hat{n} = \underset{n \in \mathbb{N}}{arg\,min} \{p_n : p_n \geq \theta\}$$

i.e., the firing will occur at the end of the $\hat{n}$-th accumulation period, which means during the $\hat{t}$-th time unit since t_0, thus:

$$\hat{t} = \hat{n} \cdot T + t_0$$

where t_0 is the *last* reset time, i.e., the last instant back in time when the neuron completed its refractory period. Then the *next* reset time t', i.e., the next instant in future when the neuron will complete its refractory period, after having emitted a spike, is:

$$t' = \hat{t} + \tau = \hat{n} \cdot T + \tau + t_0$$

At instant t', the neuron quits its refractory period, n is reset to 0, t_0 is set to t', and $\hat{n}$, $\hat{t}$ and t' must be consequently re-computed as described above.

Such a way to describe our model dynamics allow us to express the *inter-firing period* as a function of $\hat{n}$:

$$t' - t_0 = \hat{n} \cdot T + \tau$$

So, the minimum inter-firing period is $T + \tau$ for $\hat{n} = 1$. □

Such a property can also be verified as follows: let $\mathcal{I}$ be the non-deterministic input generator having $T_{min} = 1$ and, without loss of generality[1], let the initial delay $D = T + \tau$. Then the timed automata network $\mathcal{I} \| \mathcal{N} \| \mathcal{O}$ satisfies the following formula:

$$AG(state_{\mathcal{O}}(\mathbf{O}) \rightarrow eval_{\mathcal{O}}(s) \geq T + \tau)$$

where s measures the time elapsed since last firing, meaning that, whenever the output consumer receives a spike, the time elapsed since the previous received spike cannot be lower than $T + \tau$.

The next fact states that only positive stimulations are necessary for the neuron to produce emissions.

Fact 1 *Let $\mathcal{N} = (\theta, \tau, \lambda, p, y)$ be a neuron, $a(t)$ the sum of weighted inputs received during the current accumulation period, and $p(t-1)$ the neuron potential at the end of the previous accumulation period. If $p(t-1) < \theta$ and $a(t) < 0$, the neuron cannot fire at the end of the current accumulation period. Moreover, if $p(t) \geq \theta$ then $a(t) > 0$.*

The neuron potential is affected by every input spike it received *since the last reset time*, but every event that occurred before that instant is forgotten: i.e., neurons are memoryless.

Definition 7 (Inter-emission memory) Let $\mathcal{N}$ be a neuron, $Z_{\mathcal{N}}$ its reset times set, and I an input sequence. Then $\mathcal{N}$ has inter-emission *memory* if and only if there exist two different $t, t' \in Z_{\mathcal{N}}$ such that the output sequences produced by $\mathcal{N}$ as a response to I starting from t and t' are different.

Property 3 Neurons have not inter-emission memory.

Proof When the neuron moves from location **W** to **A**, it resets clock t and variables p and a, making them equal to their initial values. This entails that the neuron, if subjected to the same input sequence, will always behave in the same way. □

Next, we validate the neuron model against its ability of reproducing or not some behaviours, as described by Izhikevich in [25]. We introduce first three behaviours that are verified by our model.

[1] the initial delay is required in order to make the formula hold for the first output spike too

Tonic Spiking. *Tonic spiking* is the behaviour of a neuron producing a periodic output sequence as a response to a persistent excitatory constant input sequence.

Property 4 (Tonic spiking) Let $\mathcal{N} = (\theta, \tau, \lambda, p, y)$ be a neuron having only one ingoing excitatory synapse of weight w and let $\mathcal{I}$ be the input source connected to $\mathcal{N}$ producing a persistent input sequence. Then $\mathcal{N}$ produces a periodic output sequence.

The property holds by construction. It can be tested via model checking in the following way. Let $\mathcal{I}$ be the fixed-rate input generator having arbitrary initial delay D, and let $\mathcal{O}$ be an output consumer. Then the timed automata network $\mathcal{I}\|\mathcal{N}\|\mathcal{O}$ satisfies the following formulae:

$$\begin{cases} state_{\mathcal{O}}(\mathbf{O}) \wedge eval_{\mathcal{O}}(e) \rightsquigarrow state_{\mathcal{O}}(\mathbf{O}) \wedge \neg eval_{\mathcal{O}}(e) \\ state_{\mathcal{O}}(\mathbf{O}) \wedge \neg eval_{\mathcal{O}}(e) \rightsquigarrow state_{\mathcal{O}}(\mathbf{O}) \wedge eval_{\mathcal{O}}(e) \end{cases}$$

where $\mathbf{O}$ is the location that the consumer automaton $\mathcal{O}$ reaches after consuming a spike and e is an alternating boolean variable whose value flips whenever $\mathcal{O}$ moves into location $\mathbf{O}$. So, whenever automaton $\mathcal{O}$ reaches location $\mathbf{O}$, it will eventually reach it again.

One may also find the value P of the period of some given neuron $\mathcal{N}$ by means of simulations, thus the periodic behaviour can be proven verifying the following formula:

$$AG(state_{\mathcal{O}}(\mathbf{O}) \wedge eval_{\mathcal{N}}(f) \rightarrow eval_{\mathcal{O}}(s) = P)$$

where s is the clock measuring the time elapsed since last spike consumed by $\mathcal{O}$, and f is a boolean variable of automaton $\mathcal{N}$ which is initially $false$ and is set to $true$ when edge $(\mathbf{W} \rightarrow \mathbf{A})$ fires (i.e., it indicates whether $\mathcal{N}$ has already emitted the first spike and waited the first refractory period or not).

Integrator. *Integrator* is the behaviour of a neuron producing an output spike whenever it receives *at least* a specific number of spikes from its input sources in the same accumulation period.

Property 5 (Integrator) Let $\mathcal{N} = (\theta, \tau, \lambda, p, y)$ be a neuron having m synapses with maximum excitatory weight R and a threshold $n \leq m$. Then the neuron emits if it receives a spike from at least n input sources during the same accumulation period.

As in the previous case, we can use model checking tools and test the formula stating that, if at least n generators are ready to emit (location $\mathbf{S}$) while $\mathcal{N}$ is in $\mathbf{A}$, then $\mathcal{O}$ will eventually capture an output of $\mathcal{N}$:

$$\left(\sum_{i=1}^{m} state_i(\mathbf{S}) \geq n \right) \wedge state_{\mathcal{N}}(\mathbf{A}) \rightsquigarrow state_{\mathcal{O}}(\mathbf{O})$$

Notice that, since potential depends on past inputs too, the neuron may still be able to fire in other circumstances, e.g., if it keeps receiving less than n spikes for a sufficient number of accumulation periods, then it may eventually fire.

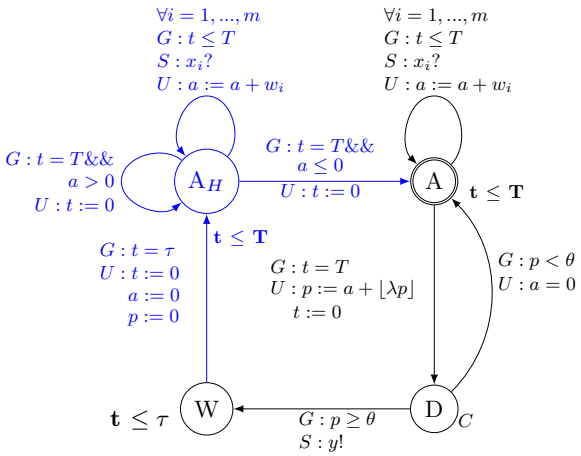


Fig. 4: The extended neuron model for phasic spiking.
Additions are colored in blue.

Excitability. *Excitability* is the behaviour of a neuron emitting sequences having a *decreasing* inter-firing period, i.e., an increasing output frequency, when stimulated by an *increasing* number of excitatory inputs.

Property 6 (Excitability) Let $\mathcal{N} = (\theta, \tau, \lambda, p, y)$ be a neuron having m excitatory synapses. Then the inter-spike period decreases as the sum of weighted input spikes increases.

Proof If we assume the neuron is receiving an increasing number of excitatory spikes, generated by an increasing number of input sources emitting persistent inputs, then a_t is the non-negative, non-decreasing and progressing (i.e., $\forall u \ \exists t : a_t > u$) succession representing the weighted sum of inputs within the t-th time unit. Consequently, $A_n = \sum_{k=1}^{T} a_{k+t_0}$ is the non-negative, non-decreasing and progressing succession counting the total sum of inputs within the n-th accumulation period. Since A_n is positive and Property 2 holds, we can prove that the inter-spike period $t_n - t_{n-1}$ decreases. □

The following behaviours are not satisfied by the LI&F model, we show that our encoding cannot verify them as well.

Phasic Spiking. *Phasic spiking* is the behaviour of a neuron producing a *single* output spike when receiving a persistent and excitatory input sequence and then remaining quiescent for the rest of it. Such a behaviour depends on the neuron to have inter-emission memory.

Property 7 Neurons cannot reproduce the *phasic spiking* behaviour.

Proof The phasic spiking behaviour requires the neuron to ignore any excitatory input spike occurring after its first emission. This means producing different outcomes, before and after the first emission, as a response to the same input sequence, which is impossible for a memoryless neuron, as stated in Property 3. □

We can extend our model to reproduce phasic spiking, see Figure 4. This variant makes the neuron able to "remember" if it is receiving a persistent excitatory input sequence. After each refractory period, the neuron moves to location $\mathbf{A_H}$, instead of $\mathbf{A}$. The only difference between $\mathbf{A_H}$ and $\mathbf{A}$ is that $\mathbf{A_H}$ *ignores* positive values of a at the end of each accumulation period. Conversely, a non-positive value of a (denoting the end of the persistent input), at the end of some accumulation period, leads the neuron back in location $\mathbf{A}$.

Bursting. A *burst* is a finite sequence of *high frequency* spikes. More formally:

Definition 8 A spike output sequence is a *burst* if it is composed by spikes having an occurrence rate greater than $1/\tau$, with τ being the refractory period of the neuron.

A *burst sequence* is a sequence composed by bursts, subject to the following constraint: the time difference between the last spike of each burst and the first spike of the next burst it greater than τ.

Property 8 Neurons cannot produce bursts.

Proof A neuron $\mathcal{N}$ cannot emit spikes having a rate greater than $1/(T+\tau)$, as stated by Property 2, so it cannot produce bursts. □

In order to reproduce bursts our model can be extended by allowing several subsequent emissions in an interval period smaller than τ. After this period all clocks and variables are reset and the accumulation-fire-rest cycle can start again.

Several bursting behaviours are described in [25]. Here we discuss only three of them, as all impossibility results depend on Property 8 and all the automata extensions are similar.

Tonic Bursting is the behaviour of a neuron producing a burst sequence as a response to a persistent and excitatory input sequence. *Phasic Bursting* is the behaviour of a neuron producing a burst as a consequence of a persistent excitatory input sequence and then remaining quiescent. Obviously the preceding behaviours require the ability of producing bursts.

Bursting-then-Spiking is the behaviour of a neuron producing a burst as response to a persistent excitatory input sequence and then producing a periodic output sequence. Such a behaviour, similarly to Phasic and Tonic Bursting, depends on the neuron ability of producing bursts. Moreover it requires inter-emission memory, in order to detect the beginning of a persistent sequence.

Property 9 Neurons cannot exhibit the *Tonic Bursting, Phasic Bursting* and *Bursting-then-Spiking* behaviours.

Proof Follows from Property 8. □

Spike Frequency Adaptation. *Spike Frequency Adaptation* is the behaviour of a neuron producing a decreasing-frequency output sequence as a response to a persistent excitatory input sequence. In other words, the inter-emission time difference increases as the time elapses. This behaviour requires the neuron to have inter-emission memory as it should be able to keep track of the time elapsed since the beginning of the input sequence.

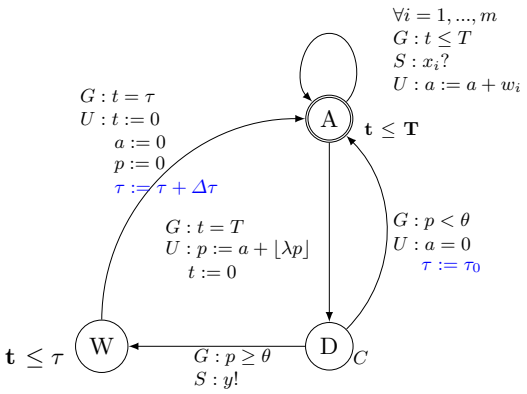


Fig. 5: The extended model for Spike Frequency Adaptation behaviour. Additions are colored in blue.

Property 10 Neurons cannot reproduce the *Spike Frequency Adaptation* behaviour.

Proof The Spike Frequency Adaptation behaviour requires the neuron to detect the beginning of an excitatory input sequence and to increase the time required to fire a spike, after each emission. This means the neuron will produce different outcomes as response to equal inputs, which is impossible, as stated in Property 3. □

An extended neuron model able to reproduce Spike Frequency Adaptation behaviour is shown in Figure 5. This variant allows the refractory period to increase after each neuron emission, thus making the output frequency decrease.

Spike Latency. *Spike Latency* is the behaviour of a neuron firing delayed spikes, with respect to the instant when its potential reached or overcame the threshold. Such a delay is proportional to the strength of the signal which leads it to emission, i.e., the sum of weighed inputs received during the accumulation period preceding the emission. This behaviour requires the neuron to be able to postpone its output.

Property 11 Neurons cannot reproduce the *Spike Latency* behaviour.

Proof The property holds by construction. As location **D** is *committed*, no firing can be delayed. □

An easy solution to extend our model is to introduce a delay between the instant the neuron reaches or overcomes its threshold and the actual emission instant. Such a delay δ depends on the sum of weighted inputs received during the last accumulation period. If the potential is greater than or equal to the threshold, the neuron computes the delay duration $\delta(a)$, assigning it to an integer variable d, and then waits in location **Del** for d time units before emitting a spike on channel y. The extended version is depicted in Figure 6.

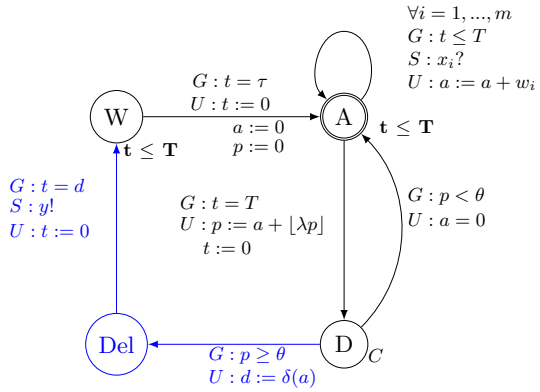


Fig. 6: The extended model for the Spike Latency behaviour. Additions are colored in blue.

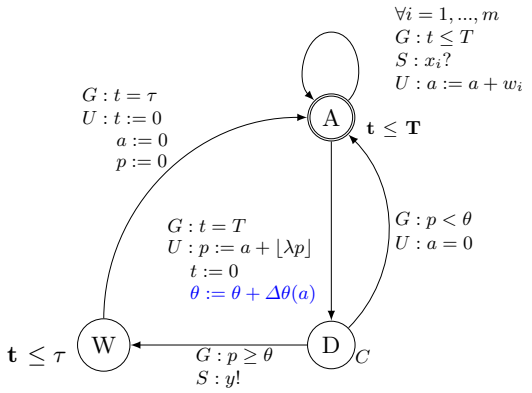


Fig. 7: The extended model for the Threshold variability behaviour. Additions colored are in blue.

Threshold Variability. *Threshold variability* is the behaviour of a neuron allowing its threshold to vary according to the strength of its inputs. More precisely, an excitatory input will rise the threshold while an inhibitory input will decrease it. As a consequence, excitatory inputs may more easily lead the neuron to fire when occurring after an inhibitory input.

Property 12 Neurons cannot reproduce the *Threshold Variability* behaviour.

Proof By construction the neuron threshold never changes. □

The neuron model can be extended allowing the threshold to vary after each accumulation period according to the current sum of weighted inputs (see Figure

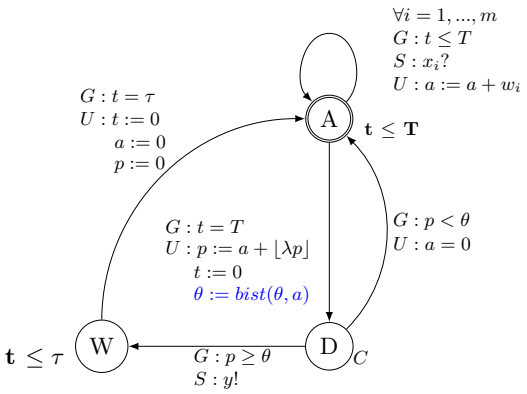


Fig. 8: The extended model for Bistability behaviour.
Additions are colored in blue.

7). The threshold variable initial value is θ_0. On every firing of edge ($\mathbf{A} \rightarrow \mathbf{D}$), the threshold variable is increased of $\Delta(a)$, where a is the sum of weighted inputs occurred during the last accumulation period and $\Delta(a)$ is an integer value whose sign is opposite to the sign of a and whose magnitude is proportional to the magnitude of a.

Bistability. *Bistability* is the behaviour of a neuron alternating between two operation modes: *periodic emission* and *quiescence*. Upon reception of a single excitatory spike, it emits a periodic output sequence and switches to a quiescent mode (no emission) as soon as it received another spike. Such a behaviour requires the neuron to (i) be able to produce a periodic output sequence, even if no excitatory spike is received, (ii) be able to remain silent when no spike is received, and (iii) be able to switch between the two operation modes upon reception of an excitatory spike.

Property 13 Neurons cannot reproduce the *Bistability* behaviour.

Proof The only possibility of obtaining a periodic output as a result of no excitatory input spike is to set $\theta = 0$. This is a limit case of Property 4. Since, by construction, the threshold cannot vary, the neuron cannot switch between the two operation modes. □

The neuron model can be modified as shown in Figure 8. This variant makes its threshold switch between 0 and a positive value at the end of any accumulation period during which it received an excitatory sum of weighted inputs a. A null threshold would make the neuron emit even if no input is received. Conversely, a positive threshold would prevent the neuron from emitting, if no input is received. Thus, on every firing of edge ($\mathbf{A} \rightarrow \mathbf{D}$), the threshold value θ is computed by the

function $bist(\cdot)$:

$$bist(\theta, a) = \begin{cases} 0 & if\ \theta > 0 \wedge a > 0 \\ \theta_0 & if\ \theta = 0 \wedge a > 0 \\ \theta & if\ a \leq 0. \end{cases}$$

Inhibition-induced activities. This is the behaviour of a neuron producing a spike output sequence as a response to a persistent *inhibitory* input sequence. We thus require the neuron to be able to emit as a consequence of some inhibitory input spikes.

Property 14 Neurons cannot reproduce the *Inhibition-induced Spiking* behavior.

Proof Follows from Fact 1. □

An easy extension to our automata is to consider *the absolute value* of all inputs instead of their signed values.

Rebound activities. *Rebound Spike* is the behaviour of a neuron producing an output spike after it received an inhibitory input. Similarly to Inhibition-induced activities, this behaviour requires the neuron to emit as a consequence of an inhibitory input spike.

Property 15 Neurons cannot exhibit the *Rebound Spiking* behaviour.

Proof Follows from Fact 1. □

We can modify our encoding by setting the neuron potential to be always non-negative and by fixing the threshold to be 0 as response to an inhibitory stimulation. Recall that a null threshold would make the neuron emit even if its potential is 0. Thus, on every firing of the edge $(\mathbf{A} \rightarrow \mathbf{D})$, if the current sum of weighted inputs a is negative, the threshold θ is set to 0, otherwise it is set to a $\theta > 0$. This will allow an inhibitory stimulus to produce a rebound spike.

5 Parameter inference

In this section we examine the *Learning Problem*: i.e., how to determine a parameter assignment for a network with a fixed topology and a given input such that a desired output behaviour is displayed. Here we only focus on the estimation of synaptic weights in a given spiking neural network; the generalisation of our methodology to other parameters is left for future work.

Our analysis takes inspiration from the SpikeProp algorithm [5], which deals with fully connected feedforward networks of spiking neurons with layers and aims at attaining a set of target firing times of the output neurons for a given set of input patterns. An error function is obtained by computing the least mean squares of desired spike times and actual firing times; such an error is then back-propagated in the network. In a similar way, we detect the errors of output neurons and back-propagate them. The main differences between the SpikeProp approach and our ones are: (i) the SpikeProp rule deals with multi-layered cycle-free continuous spiking neural networks while our networks are discrete and not multi-layered; (ii)

Algorithm 1 The advice back-propagation algorithm

```
1: function ABP
2:     discovered = ∅
3:     for all 𝒩 ∈ Output do
4:         if 𝒩 ∉ discovered then
5:             discovered = discovered ∪ 𝒩
6:             if Evaluate(𝒩) = SHF then
7:                 SHF(𝒩)
8:             else if Evaluate(𝒩) = SNHF then
9:                 SNHF(𝒩)
```

the Spikeprop rule does not treat negative edges (for this, inhibitory and excitatory neurons are introduced) while our connections are allowed to contain a mix of both positive and negative edges.

In the technique we propose here, the learning process is led by *supervisors*. Differently from the previous section, each output neuron $\mathcal{N}$ is linked to a supervisor instead of an output consumer. Supervisors compare the expected output behaviour with the one of the output neuron they are connected to (function Evaluate($\mathcal{N}$) in Algorithm 1). Thus either the output neuron behaved consistently or not. In the second case and in order to instruct the network, the supervisor back-propagates *advices* to the output neuron depending on two possible scenarios: i) the neuron fires a spike, but it was supposed to be quiescent, ii) the neuron remains quiescent, but it was supposed to fire a spike. In the first case the supervisor addresses a *should not have fired* message (SNHF) and in the second one a *should have fired* (SHF). Then each output neuron modifies its ingoing synaptic weights and in turn behaves as a supervisor with respect to its predecessors, back-propagating the proper advice.

The advice back-propagation (ABP), Algorithm 1, basically lies on a depth-first visit of the graph topology of the network. Let $\mathcal{N}_i$ be the i-th predecessor of an automaton $\mathcal{N}$, then we say that $\mathcal{N}_i$ fired, if it emitted a spike during the current or previous accumulate-fire-wait cycle of $\mathcal{N}$. Thus, upon reception of a SHF message, $\mathcal{N}$ has to *strengthen* the weight of each ingoing *excitatory* synapse and *weaken* the weight of each ingoing *inhibitory* synapse. Then, it propagates a SHF advice to each ingoing *excitatory* synapse (i.e., an arc with weight greater than 0: $\mathrm{Wt} \geq 0$) corresponding to a neuron which *did not* fire recently ($\neg\mathrm{F}(\mathcal{N})$), and symmetrically a SNHF advice to each ingoing *inhibitory* synapse ($\mathrm{Wt} < 0$) corresponding to a neuron which fired recently (see Algorithm 2 for SHF, and Algorithm 3 for the dual case of SNHF). When the graph visit reaches an input generator, it will simply ignore any received advice (because input sequences should not be affected by the learning process). The learning process ends when all supervisors do not detect any more errors.

There are several possibilities on how to realise supervisors and the ABP algorithm. We propose here two approaches. The first one is model checking oriented and it is based on the idea that supervisors are represented by temporal logic formulae. The second one is simulation oriented, and the implementation of the algorithm is embedded into the timed automata modelling of the neuron.

Algorithm 2 Should Have Fired algorithm

```
1: procedure SHOULD-HAVE-FIRED(𝒩)
2:     if 𝒩 ∈ discovered ∪ Output then
3:         return
4:     discovered = discovered ∪ 𝒩
5:     for all ℳ ∈ PRED(𝒩) do
6:         if ℳ ∉ Input then
7:             if WT(ℳ,𝒩) ≥ 0 ∧ ¬ F(ℳ) then
8:                 SHF(ℳ)
9:             if WT(ℳ,𝒩) < 0 ∧ F(ℳ) then
10:                SNHF(ℳ)
11:        INCREASE-WEIGHT(ℳ,𝒩)
12:    return
```

Algorithm 3 Abstract ABP: Should *Not* Have Fired advice pseudo-code

```
1: procedure SHOULD-NOT-HAVE-FIRED(neuron)
2:     if 𝒩 ∈ discovered ∪ Output then
3:         return
4:     discovered = discovered ∪ 𝒩
5:     for all ℳ ∈ PREDECESSORS(𝒩) do
6:         if ℳ ∉ Input then
7:             if WT(ℳ,𝒩) ≥ 0 ∧ F(ℳ) then
8:                 SNHF(ℳ)
9:             if WT(ℳ,𝒩) < 0 ∧ ¬ F(ℳ) then
10:                SHF(ℳ)
11:        DECREASE-WEIGHT(ℳ,𝒩)
12:    return
```

Model-checking-oriented approach. Such a technique consists in iterating the learning process until a desired CTL property concerning the output of the network is verified. The hypothesis we introduce are the following ones: (i) input generators, standard neurons, and output consumers share a global clock which is never reset and (ii) for each output consumer, there exists a clock measuring the elapsed time since the last received spike. The CTL formula specifying the expected output of the network can only contain predicates relative to the output consumers and the global clock. At each step of the algorithm, we make an external call to the model checker to test whether the network satisfies the formula or not. If the formula is verified, the learning process ends; otherwise, the model checker provides a trace as a counterexample. Such a trace is exploited to derive the proper corrective action to be applied to each output neuron, that is, the invocation of either the SHF procedure, or the SNHF procedure previously described (or no procedure).

More in detail, given a timed automata network representing some spiking neural network, we extend it with a global clock t_g which is never reset and, for each output consumer $\mathcal{O}_K$ relative to the output neuron $\mathcal{N}_k$, we add a clock s_k measuring the time elapsed since the last spike consumed by $\mathcal{O}_k$. Furthermore, let

$state_{\mathcal{O}_k}(\mathbf{O})$ be an atomic proposition evaluating to true if the output consumer $\mathcal{O}_K$ is in its $\mathbf{O}$ location, and let $eval_{\mathcal{O}_k}(s_k)$ be an atomic proposition indicating the value of the clock s_k in $\mathcal{O}_K$. In order to make it possible to deduce the proper corrective action, we impose the CTL formula describing the expected outcome of the network to be composed by the conjunction of sub-formulae respecting any of the patterns presented in the following.

Precise Firing. The output neuron $\mathcal{N}_k$ fires at time t:
$AF\,(\,t_g = t \wedge state_{\mathcal{O}_k}(\mathbf{O})\,)$.
The violation of such a formula requires the invocation of the SHF procedure.
Weak Quiescence. The output neuron $\mathcal{N}_k$ is quiescent at time t:
$AG\,(\,t_g = t \implies \neg state_{\mathcal{O}_k}(\mathbf{O})\,)$.
The SNHF procedure is called in case this formula is not satisfied.
Relaxed Firing. The output neuron $\mathcal{N}_k$ fires at least once within the time window $[\,t_1, t_2\,]$:
$AF\,(\,t_1 \leq t_g \leq t_2 \wedge state_{\mathcal{O}_k}(\mathbf{O})\,)$.
The violation of such a formula leads to the invocation of the SHF procedure.
Strong Quiescence. The output neuron $\mathcal{N}_k$ is quiescent for the whole duration of the time window $[\,t_1, t_2\,]$:
$AG\,(\,t_1 \leq t_g \leq t_2 \implies \neg state_{\mathcal{O}_k}(\mathbf{O})\,)$.
The SNHF procedure is needed in this case.
Precise Periodicity. The output neuron $\mathcal{N}_k$ eventually starts to periodically fire a spike with exact period P:
$AF(\,AG(\,eval_{\mathcal{O}_k}(s_k) \neq P \implies \neg state_{\mathcal{O}_k}(\mathbf{O})\,)$
$\wedge\, AG(\,state_{\mathcal{O}_k}(\mathbf{O}) \implies eval_{\mathcal{O}_k}(s_k) = P\,)\,)$.
If $\mathcal{N}_k$ fires a spike while the s_k clock is different than P or it does not fire a spike while the s_k clock equals P, the formula is not satisfied. In the former (resp. latter) case, we deduce that the SNHF (resp. SHF) procedure is required.
Relaxed Periodicity. The output neuron $\mathcal{N}_k$ eventually begins to periodically fire a spike with a period that may vary in $[\,P_{min}, P_{max}\,]$:
$AF(\,AG(\,eval_{\mathcal{O}_k}(s_k) \notin [\,P_{min}, P_{max}\,] \implies$
$\neg state_{\mathcal{O}_k}(\mathbf{O})\,) \wedge$
$AF(\,state_{\mathcal{O}_k}(\mathbf{O}) \implies$
$P_{min} \leq eval_{\mathcal{O}_k}(s_k) \leq P_{max}\,)\,)$.
For the corrective actions, see the previous case.

As for future work, we intend to extend this set of CTL formulae with new formulae concerning the comparison of the output of two or more given neurons. Please notice that the Uppaal model-checker only supports a fragment of CTL where the use of nested path quantifiers is not allowed. Another model-checker should be called in order to fully exploit the expressive power of CTL.

Simulation-oriented approach. In this second approach, parameters are modified during the simulation of the network. This entails that the encoding of neurons needs to be adjusted in order to take care of the adaptation of such parameters. Algorithm ABP is realised by a dedicated automaton $ABP - alg$, that is depicted in Figure 9, and the role of supervisor is given to output consumers, that are modified as in Figure 10.

The idea is that, according to the function EVALUATE, if the corresponding output neuron misbehaves, then its output consumer sets whether it has to be

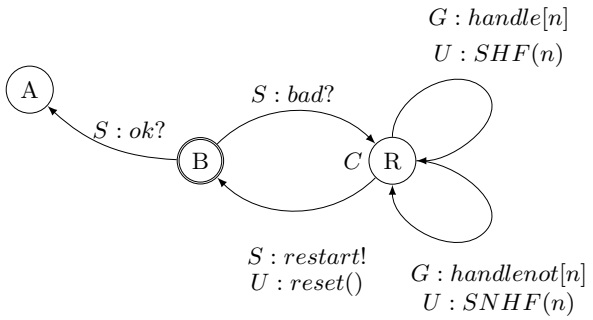


Fig. 9: The automata responsible of the $ABP - alg$

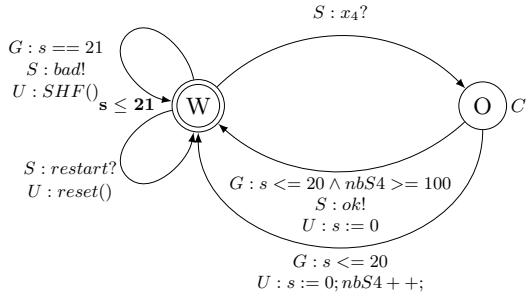


Fig. 10: Example of an output consumer in the simulation approach for the diamond structure

treated according to the SHF or the SNHF function. Furthermore, it signals to the $ABP - alg$ through the message *bad*! that some adjustments on the network have to be done. Then the $ABP - alg$ automaton takes the lead and it recursively applies the function SHF or SNHF (this is achieved by setting a proper variable in a vector named *handle*) on the predecessors of the output neuron. Once there is no more neuron to whom the algorithm should be applied (for instance all neurons in the current run have been visited), the simulation is restarted in the network with the new parameters. If the output consumer does not recognise any misbehaviour, than it sends an *ok*! message to the $ABP - alg$ automaton, that in turn moves to an accepting state A.

More formally:

Definition 9 (Output consumer for the simulation approach) An output consumer for the simulation approach is a timed automaton

$$\mathcal{N} = (L, \mathbf{A}, X, Var, \Sigma, Arcs, Inv)$$

with:

- $L = \{\mathbf{W}, \mathbf{O}\}$ with $\mathbf{O}$ committed,
- $X = \{s\}$
- $Var = \{nb, handle[N], handlenot[N]\}$
- $\Sigma = \{x_i \mid x_i \text{ is an output neuron}\} \cup \{bad, restart, ok\}$,
- $Arcs = \{(\mathbf{W}, s < T_{min}, bad!, \{SHF()\}, \mathbf{W})$,
  $(\mathbf{W}, , restart?, \{s := 0, nb := 0\}, \mathbf{W})$,
  $(\mathbf{W}, , x_i?, , \mathbf{0}), (\mathbf{O}, s > T_{max}, bad!, \{SNHF()\}, \mathbf{W})$,
  $(\mathbf{O}, , restart?, \{s := 0, nb := 0\}, \mathbf{O})$,
  $(\mathbf{O}, good\_pattern, ok!, \{s := 0\}, \mathbf{W})$,
  $(\mathbf{O}, good\_not\_finished, , \{s := 0\}, \mathbf{W})\}$;
- $Inv(\mathbf{W}) = s \leq T$, $Inv(\mathbf{O}) = \mathbf{true}$

where the functions SHF and SNHF modify the global variables $handle_i$ and $handlenot_i$ respectively, that are used in the $ABP - alg$ automaton.

Notice that the precise definition (for instance the value of parameters $good\_pattern$, $good\_not\_finished$, T_{min}, T_{max}) of the output consumer depends on the expected behaviour of the supervisor. As an example, in Figure 10, that depicts the output consumer for the diamond network (see the details in the next subsection), we expect the output neuron to emit a spike within each window of 20 seconds for at least 100 times.

More in detail, the cycle from $\mathbf{W}$ to $\mathbf{W}$ is taken whenever a spike has not been sent before T time units. Thus the automaton sends to the $ABP - alg$ automaton a *bad* message (signalling that an adjustment to the network should take place), and the update part handles the fact that we should perform the SHF algorithm (a variable corresponding to the concerned output neuron is set to **true** in the array *handle*). From the same state, whenever the message *restart* is received, all the variables of the automaton are reset to 0. When a spike from the output neuron x is received, the output consumer moves to the state $\mathbf{O}$. In this state, if the spike was received too late (and similarly as in the previous case), a *bad* message is sent to the $ABP - alg$. Otherwise, if everything was received on time and if the expected pattern has been completely verified, then an *ok* message is sent and the automaton moves to the initial state.

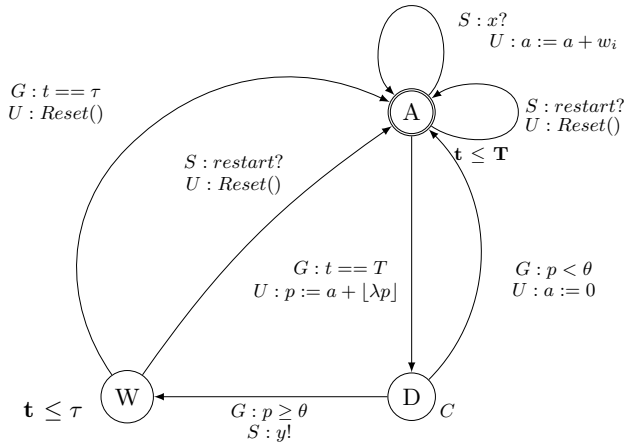
In the following, we give the formal definition of the $ABP - alg$ automaton:

Definition 10 ($ABP - alg$ **automaton)**

- $L = \{\mathbf{A}, \mathbf{B}, \mathbf{R}\}$ with $\mathbf{R}$ committed,
- $X = \emptyset$
- $Var = \{handle[N], handlenot[N]\}$
- $\Sigma = \{bad, ok, restart\}$,
- $Arcs = \{(\mathbf{B}, , ok?, , \mathbf{A}), (\mathbf{B}, , bad?, , \mathbf{R})$
  $(\mathbf{R}, handle[n], , SHF(n), \mathbf{R})$
  $(\mathbf{R}, handlenot[n], , SNHF(n), \mathbf{R})$
  $(\mathbf{R}, finished?(), restart!, reset(), \mathbf{B})\}$;
- $Inv(\mathbf{A}) = \mathbf{true}$, $Inv(\mathbf{B}) = \mathbf{true}$, $Inv(\mathbf{R}) = \mathbf{true}$.

The arc from the state $\mathbf{B}$ to the accepting state $\mathbf{A}$ is taken whenever the Output consumer has finished the analysis of its pattern. Conversely, the arc from the state $\mathbf{B}$ to $\mathbf{R}$ is adopted when one of the output neurons has misbehaved (signalled by the reception of the message *bad*). In the committed state $\mathbf{R}$, the parameters of

Fig. 11: Example of a neuron in the simulation approach for the diamond network



the neural network are changed accordingly to the Algorithms 2 and 2. The arrays *handle* (respectively *handlenot*) have the information on the neurons to which the Algorithm for SHF (respectively SNHF) has to be applied. Once the cycle of updates finishes (checked through the function $finished?()$), the simulation is restarted by broadcasting a message *restart* to all the neurons and the output consumer.

Last, we give the definition of the changes induced in the standard neuron. As the update of the neuron parameters is done at the level of $ABP - alg$, the only change concerns the treatment of the signal *restart*. To this aim an arc handling the reception of the message is added to the states **W** and **A**.

Definition 11 (Standard Neuron for the simulation approach) Given a neuron $v = (\theta, \tau, \lambda, p, y)$ with m input synapses, its encoding into timed automata is $\mathcal{N} = (L, \mathbf{A}, X, Var, \Sigma, Arcs, Inv)$ with:

- $L = \{\mathbf{A}, \mathbf{W}, \mathbf{D}\}$ with **D** committed,
- $X = \{t\}$
- $Var = \{p, a\}$
- $\Sigma = \{x_i \mid i \in [1..m]\} \cup \{y\}$,
- $Arcs = \{(\mathbf{A}, t \leq T, x_i?, \{a := a + w_i\}, \mathbf{A}) \mid i \in [1..m]\} \cup \{(\mathbf{A}, t = T, \ , \{p := a + \lfloor \lambda p \rfloor\}, \mathbf{D}),$
  $(\mathbf{D}, p < \theta, \ , \{a := 0\}, \mathbf{A}), (\mathbf{D}, p \geq \theta, y!, \ , \mathbf{W}),$
  $(\mathbf{W}, t = \tau, \ , \{a := 0, t := 0, p := 0\}, \mathbf{A}),$
  $(\mathbf{A}, , restart?, \{a" = 0, t := 0, p := 0\}\mathbf{A}),$
  $(\mathbf{W}, , restart?, \{a" = 0, t := 0, p := 0\}\mathbf{A})\}$;
- $Inv(\mathbf{A}) = t \leq T, Inv(\mathbf{W}) = t \leq \tau, Inv(\mathbf{D}) = \mathbf{true}$.

Notice that the algorithm can be refined by setting some priorities on the neurons. For instance, if any additional information is known in advance on the behaviour of a specific neuron, its actions can be constrained and the corrective

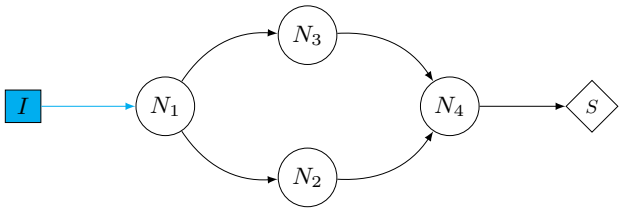


Fig. 12: A neural network with a diamond structure.

operations could be ignored. In some cases, the type of synapse (excitatory or inhibitory) can also be set, disallowing the possibility of changing its nature (e.g., from inhibitory to excitatory).

Notice that, as the application of the ABP algorithm in the simulation approach is non-deterministic, the parameters we found may depend on the precise execution and several solutions are therefore possible.

5.1 Examples

In this section we show several examples where we have applied our approaches to determine a parameter assignment. The complete encoding of the examples shown here can be found at [13].

Turning on and off a diamond structure of neurons. This example shows how the ABP algorithm can be used to make a neuron emit at least once in a spiking neural network having the *diamond* structure shown in Figure 12. We assume that $\mathcal{N}_1$ is fed by an input generator $\mathcal{I}$ that continuously emits spikes. No neuron in the network is able to emit because all the weights of their input synapses are equal to zero and their thresholds are higher than zero. The initial weights and parameters are:

$w_{0,1}$	$w_{1,2}$	$w_{1,3}$	$w_{2,4}$	$w_{3,4}$
0.1	0.1	0.1	0.1	0.1

Neuron	T	θ	τ	λ
$\mathcal{N}_1$	2	0.35	3	7/9
$\mathcal{N}_2$	2	0.35	3	7/9
$\mathcal{N}_3$	2	0.35	3	7/9
$\mathcal{N}_4$	2	0.55	3	1/2

We want the network to learn a weight assignment so that $\mathcal{N}_4$ is able to emit, that is, to produce a spike after an initial pause.

At the beginning we expect no activity from neuron $\mathcal{N}_4$. As soon as the initial pause is elapsed, we require a spike but, as all weights are equal to zero, no emission can happen. Thus a SHF advice is back-propagated to neurons $\mathcal{N}_2$ and $\mathcal{N}_3$ and consequently to $\mathcal{N}_1$. The process is then repeated until all weights stabilise and neuron $\mathcal{N}_4$ is able to fire.

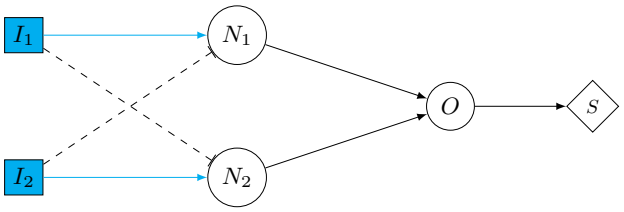


Fig. 13: Structure of the XOR network. Dashed (resp. continuous) edges stand for inhibitions (resp. activations).

Here we apply the *model-checking approach*. We expect $\mathcal{N}_4$ to spike every 20 time units. After three cycles of the algorithm (i.e., three checks of the formula and modifications of weights), we reach the following weight assignment:

$w_{0,1}$	$w_{1,2}$	$w_{1,3}$	$w_{2,4}$	$w_{3,4}$
0.3	0.3	0.3	0.3	0.3

The XOR problem. The XOR problem is a classic problem in neural networks. It is used to classify inputs interpreting the outputs as the result of the logic XOR between the inputs (a logic XOR returns true if the two inputs are not equal and false otherwise). The problem is interesting because it can be solved only within a multilayered architecture. Figure 13 shows an example of such structure. Although typical in neural networks, this example is not adapted to our setting. Our networks expect as input streams of spikes and, as mentioned in the introduction, we do not aim at classifying specific inputs.

To adjust the XOR problem to our setting, we consider a network where neurons have no memory (the leak factor is 0) and the refractory period is also 0. As input we consider two finite sequences $I_1 = P(8).s.P(3).s$ and $I_2 = P(4).s.P(7).s$ encoding the sequence of bits 0 0 1 1 for I_1 and 0 1 0 1 for I_2. The expected output is $S = P(5).s.P(4).s$ encoding the sequence of bits 0 1 1 0. Figure 14 shows the realisation of the output consumer. The other parameters are $T = 1$ and $\Theta = 1$ while the weights are

$w_{I_1,1}$	$w_{I_1,2}$	$w_{I_2,1}$	$w_{I_2,2}$	$w_{1,O}$	$w_{2,O}$
0.1	−0.1	0.1	−0.1	0.1	0.1

With the *simulation approach* we reach the following assignment:

$w_{I_1,1}$	$w_{I_1,2}$	$w_{I_2,1}$	$w_{I_2,2}$	$w_{1,O}$	$w_{2,O}$
1	−0.1	1	−0.1	1	1

A negative loop. In this example, we show a network that is meant to oscillate and we use our learning algorithm to find parameters for a specific oscillation period. The network is showed in Figure 15, it consists of two neurons N_1 and N_2: the first one excites the second one while the second one inhibits the first one. We expect N_2 to fire each 3 time units.

In this example we use *a mixture of the two approaches* described above. We model the network following the simulation approach and we check the formula

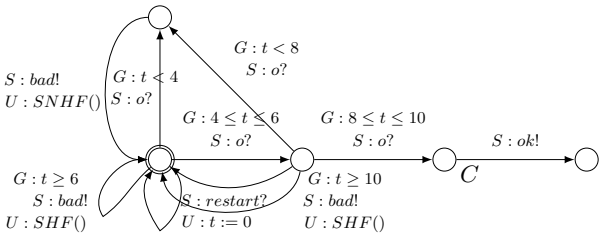


Fig. 14: The output consumer for the XOR network.

Fig. 15: A simple negative loop network.

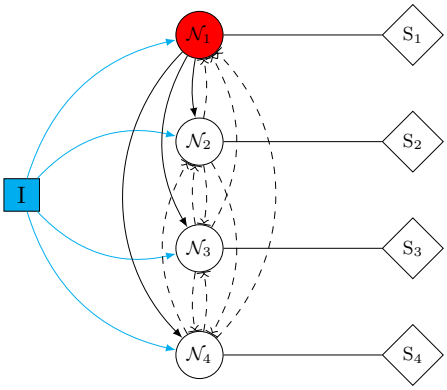


Fig. 16: We denote neurons by $\mathcal{N}_i$. The network is fed by an input generator $\mathcal{I}$ and the learning process is led by the supervisors $\mathcal{S}_i$.

$AG\neg state_{\mathcal{ABP}}(\mathbf{A})$ stating that it is not possible to reach the accepting state in the ABP automaton. The formula, as expected, is false and the counterexample provides a weight assignment.

For both neurons, we have $T = 1$, $\theta = 1$, $\lambda = 1$, and no refractory period. The initial weights are:

$w_{I,1}$	$w_{1,2}$	$w_{2,1}$
0.3	0.5	-0.7

and the counterexample provides the following weights:

$w_{I,1}$	$w_{1,2}$	$w_{2,1}$
0.5	1	-0.5

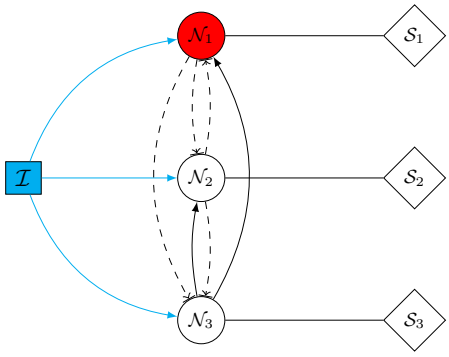


Fig. 17: A second neural network with "winner takes all" behaviour

Mutual inhibition networks. The last example is about mutual inhibition networks, where the constituent neurons inhibit each other neuron's activity. These networks belong to the set of Control Path Generators (CPGs), which are known for their capability to produce rhythmic patterns of neural activity without receiving rhythmic inputs [23]. CPGs underlie many fundamental rhythmic activities such as digesting, breathing, and chewing. They are also crucial building blocks for the locomotor neural circuits both in invertebrate and vertebrate animals. It has been observed that, for suitable parameter values, mutual inhibition networks present a behaviour of the kind "winner takes all", that is, at a certain time one neuron becomes (and stays) activated and the other ones are inhibited [14]. These networks are similar to the continuous switch networks used in modelling of the development [7].

We consider a mutual inhibition network of four neurons, as shown in Figure 16. This example, although being small, it is not trivial as it features inhibitor and excitatory edges as well as cycles.

We look for synaptical weights such that the "winner takes all" behaviour is displayed. We assume each neuron to be fed by an input generator $\mathcal{I}$ that continuously emits spikes. At the beginning, all the neurons have the same parameters (that is, firing threshold, remaining coefficient, accumulation period, and refractory period), and the weight of excitatory (resp. inhibitory) edges is set to 1 (resp. -1). We use the ABP algorithm with the *model checking approach* to learn a weight assignment so that the first neuron is the winner. More precisely, we find a weight assignment such that, whatever the chosen path in the corresponding automata network is, the network stabilises when the global clock t_g equals 70. The weight of the edges from the input generator $\mathcal{I}$ to the four neurons equals 0.041. The weight of the edges inhibiting $\mathcal{N}_1$ (resp. $\mathcal{N}_2$, $\mathcal{N}_3$, and $\mathcal{N}_4$) is -0.719 (resp. -0.817).

For the *simulation approach*, we used a smaller network with 3 neurons only, depicted in Figure 17. As before, we require a "winner takes all" behaviour where $\mathcal{N}_1$ is the winner. We expect that $\mathcal{N}_1$ spikes every 10 time units at most, and the neurons $\mathcal{N}_2$ and $\mathcal{N}_3$ do not spike. We consider a network with the following initial parameters:

Neuron	T	θ	τ	λ
$\mathcal{N}_1$	2	0.75	3	1/2
$\mathcal{N}_2$	2	0.75	3	1/2
$\mathcal{N}_3$	2	0.75	3	1/2

$w_{x,y}$	$\mathcal{I}$	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$
$\mathcal{I}$	0	0.1	0.1	0.1
$\mathcal{N}_1$	0	0	-0.1	-0.1
$\mathcal{N}_2$	0	-0.1	0	-0.1
$\mathcal{N}_3$	0	0.1	0.1	0

We obtain the following weights in 5 executions of the ABP algorithm:

$w_{x,y}$	$\mathcal{I}$	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$
$\mathcal{I}$	0	0.6	0.1	0.1
$\mathcal{N}_1$	0	0	-0.1	-0.1
$\mathcal{N}_2$	0	-0.1	0	-0.1
$\mathcal{N}_3$	0	0.6	0.1	0

6 Related Work

To the best of our knowledge, there are few attempts of giving formal models for LI&F. Apart from the already discussed approach of [14], where the authors model and verify LI&F networks thanks to the synchronous language Lustre, the closest related work we are aware of is [3]. In this work, the authors propose a mapping of spiking neural P systems into timed automata. The modelling is substantially different from ours. They consider neurons as static objects and the dynamics is given in terms of evolution rules while for us the dynamics is intrinsic to the modelling of the neuron. This, for instance, entails that inhibitions are not just negative weights as in our case, but are represented as *forgetting rules*. On top of this, the notion of time is also different: while they consider durations in terms of number of applied rules, we have an explicit notion of duration given in terms of accumulation and refractory period.

As far as our parameter learning approach is concerned, we borrow inspiration from the SpikeProp rule [5], a variant of the well known back-propagation algorithm [33] used for supervised learning in second generation learning. The SpikeProp rule deals with multi-layered cycle-free spiking neural networks and aims at training networks to produce a given output sequence for each class of input sequences. The main difference with respect to our approach is that we are considering here a discrete model and our networks are not multi-layered. We also rest on Hebb's learning rule [20] and its time-dependent generalisation rule, the spike timing dependent plasticity (STDP) rule [34], which aims at adjusting the synaptical weights of a network according to the time occurrences of input and output spikes of neurons. It acts locally, with respect to each neuron, i.e., no prior assumption on the network topology is required in order to compute the weight variations for some neuron input synapses. Differently from the STDP, our approach takes into account not only recent spikes but also some external feedback (*advices*) in order to determine which weights should be modified and whether they must increase or decrease. Moreover, we do not prevent excitatory synapses from

becoming inhibitory (or vice versa), which is usually a constraint for STDP implementations. A general overview on spiking neural network learning approaches and open problems in this context can be found in [18].

7 Conclusion

In this work timed automata networks are proposed as an effective tool to model spiking neural networks. More precisely, we focus on LI&F networks. The model we describe has been fully implemented in `Uppaal` and is available at the pages [6] and [13]. In our modeling framework, we take into account exact spike emission times of neurons rather than spike rates. The formalism we selected turned out to be very suited to model spiking neural networks: as a matter of fact, timed automata allow to model with precision several time-related aspects, such as the exact spike emission occurrences and the refractory period, a time interval which follows the spike emission and is characterised by a restricted emission capability.

In this work, model checking techniques are used to automatically prove our mapping of LI&F networks into timed automata is able to display a certain number of expected behaviours (i.e., typical responses to an input pattern), that is, tonic spiking, excitability, and integrator. Formal verification turned out to be very suited to validate our modelling framework. As for future work concerning the modeling aspects, we intend to propose automaton-based formalisations for more sophisticated spiking neuron models, such as the theta-neuron model [15] or the Izhikevich model [24]. We also plan to enrich our model with propagation delays, which seem to play an important role in spiking neural networks [30]. At this aim, we intend to add new states and clocks to model synapses. Finally, we intend to identify the neuron parameters which influence most the satisfaction of some crucial temporal logic properties. For this, a robustness analysis of the obtained model will be performed.

As key contribution, we proposed a novel technique to infer the synaptical weights of spiking neural networks. At this aim, we adapted machine learning techniques to bio-inspired models, which makes our work original and complementary with respect to the main international projects aiming at understanding the human brain, such as the Human Brain Project [10], which mainly relies on large-scale simulations.

For our learning approach, we considered a basic kind of supervisors: each supervisor only focusses on the output of a single neuron, neglecting the other neurons. However, observe that the algorithm we propose still holds when supervisors compare the output of several neurons. As for future work, we plan to encode more complex supervisors which consider the behaviour of groups of neurons. Additionally, to enhance our learning technique, we could adapt some interesting results coming from the gene regulatory network area, where some necessary and sufficient conditions linking the structure of the network and its dynamics are given [32].

To conclude, we program to generalise the ABP algorithm to infer some new parameters of neural networks, such as the leak factor or the firing threshold.

Acknowledgements We are grateful to Giovanni Ciatto for his preliminary implementation work and for his enthusiasm in collaborating with us.

References

1. Ackley, D.H., Hinton, G.E., Sejnowski, T.J.: A learning algorithm for boltzmann machines. In: D. Waltz, J.A. Feldman (eds.) Connectionist Models and Their Implications: Readings from Cognitive Science, pp. 285–307. Ablex Publishing Corp., Norwood, NJ, USA (1988)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
3. Aman, B., Ciobanu, G.: Modelling and verification of weighted spiking neural systems. Theoretical Computer Science **623**, 92 – 102 (2016). DOI http://dx.doi.org/10.1016/j.tcs.2015.11.005. URL http://www.sciencedirect.com/science/article/pii/S0304397515009792
4. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Uppaal — a Tool Suite for Automatic Verification of Real–Time Systems. In: Proceedings of Workshop on Verification and Control of Hybrid Systems III, no. 1066 in Lecture Notes in Computer Science, pp. 232–243. Springer–Verlag (1995)
5. Bohte, S.M., Poutré, H.A.L., Kok, J.N., La, H.A., Joost, P., Kok, N.: Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing **48**, 17–37 (2002)
6. Ciatto, G., De Maria, E., Di Giusto, C.: Additional material. https://github.com/gciatto/snn_as_ta (2016)
7. Cinquin, O., Demongeot, J.: High-dimensional switches and the modelling of cellular differentiation. Journal of theoretical biology **233**(3), 391–411 (2005)
8. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge, MA, USA (1999)
9. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems **2**(4), 303–314 (1989)
10. D'Angelo, E., Danese, G., Florimbi, G., Leporati, F., Majani, A., Masoli, S., Solinas, S., Torti, E.: The human brain project: High performance computing for brain cells hw/sw simulation and understanding. In: 2015 Euromicro Conference on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28, 2015, pp. 740–747. IEEE Computer Society (2015). DOI 10.1109/DSD.2015.80. URL https://doi.org/10.1109/DSD.2015.80
11. De Maria, E., Di Giusto, C.: Parameter learning for spiking neural networks modelled as timed automata. In: P. Anderson, H. Gamboa, A.L.N. Fred, S.B. i Badia (eds.) Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2018) - Volume 3: BIOINFORMATICS, Funchal, Madeira, Portugal, January 19-21, 2018., pp. 17–28. SciTePress (2018). DOI 10.5220/0006530300170028. URL https://doi.org/10.5220/0006530300170028
12. De Maria, E., Di Giusto, C., Ciatto, G.: Formal validation of neural networks as timed automata. In: Proceedings of the 8th International Conference on Computational Systems-Biology and Bioinformatics, Nha Trang City, Viet Nam, December 7-8, 2017, pp. 15–22. ACM (2017). DOI 10.1145/3156346.3156350. URL http://doi.acm.org/10.1145/3156346.3156350
13. De Maria, E., Di Giusto, C., Laversa, L.: Additional material. https://digiusto.bitbucket.io/ (2017)
14. De Maria, E., Muzy, A., Gaffé, D., Ressouche, A., Grammont, F.: Verification of Temporal Properties of Neuronal Archetypes Using Synchronous Models. In: Fifth International Workshop on Hybrid Systems Biology. Grenoble, France (2016)
15. Ermentrout, G.B., Kopell, N.: Parabolic bursting in an excitable system coupled with a slow oscillation. SIAM Journal on Applied Mathematics **46**(2), 233–253 (1986)
16. Freund, Y., Schapire, R.E.: Large margin classification using the perceptron algorithm. Machine Learning **37**(3), 277–296 (1999)
17. Gerstner, W., Kistler, W.: Spiking Neuron Models: An Introduction. Cambridge University Press, New York, NY, USA (2002)
18. Grüning, A., Bohte, S.: Spiking neural networks: Principles and challenges (2014)
19. Gruber, H., Holzer, M., Kiehn, A., König, B.: On timed automata with discrete time - structural and language theoretical characterization. In: Developments in Language Theory, 9th International Conference, DLT 2005, Palermo, Italy, July 4-8, 2005, Proceedings, pp. 272–283 (2005). DOI 10.1007/11505877_24
20. Hebb, D.O.: The Organization of Behavior. John Wiley (1949)
21. Hodgkin, A.L., Huxley, A.F.: A quantitative description of membrane current and its application to conduction and excitation in nerve. The Journal of Physiology **117**(4), 500–544 (1952)

22. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. In: J.A. Anderson, E. Rosenfeld (eds.) Neurocomputing: Foundations of Research, pp. 457–464. MIT Press, Cambridge, MA, USA (1988)
23. Ijspeert, A.J.: Central pattern generators for locomotion control in animals and robots: A review. Neural Networks **21**(4), 642–653 (2008). DOI 10.1016/j.neunet.2008.03.014. URL http://dx.doi.org/10.1016/j.neunet.2008.03.014
24. Izhikevich, E.M.: Simple model of spiking neurons. Trans. Neur. Netw. **14**(6), 1569–1572 (2003)
25. Izhikevich, E.M.: Which model to use for cortical spiking neurons? IEEE Transactions on Neural Networks **15**(5), 1063–1070 (2004)
26. Lapicque, L.: Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. J Physiol Pathol Gen **9**, 620–635 (1907)
27. Maass, W.: Networks of spiking neurons: The third generation of neural network models. Neural Networks **10**(9), 1659 – 1671 (1997)
28. Matsuoka, K.: Mechanisms of frequency and pattern control in the neural rhythm generators. Biological cybernetics **56**(5-6), 345–353 (1987)
29. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**(4), 115–133 (1943)
30. Paugam-Moisy, H., Bohte, S.: Computing with Spiking Neuron Networks, pp. 335–376. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
31. Recce, M.: Encoding information in neuronal activity. In: W. Maass, C.M. Bishop (eds.) Pulsed Neural Networks, pp. 111–131. MIT Press, Cambridge, MA, USA (1999)
32. Richard, A.: Negative circuits and sustained oscillations in asynchronous automata networks. Advances in Applied Mathematics **44**(4), 378–392 (2010)
33. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. In: J.A. Anderson, E. Rosenfeld (eds.) Neurocomputing: Foundations of Research, pp. 696–699. MIT Press, Cambridge, MA, USA (1988)
34. Sjöström, J., Gerstner, W.: Spike-timing dependent plasticity. Scholarpedia **5**(2) (2010)

Chapter 6

On the Use of Formal Methods to Model and Verify Neuronal Archetypes

Elisabetta De Maria, Abdorrahim Bahrami,Thibaud L'Yvonnet, Amy Felty, Daniel Gaffé, Annie Ressouche, and Franck Grammont

Extended version of the conference papers published at HSB 2016, CSBio 2017, and CSBio 2018

RESEARCH ARTICLE

On the Use of Formal Methods to Model and Verify Neuronal Archetypes

Elisabetta, DE MARIA[1]; Abdorrahim, BAHRAMI[2]; Thibaud, L'YVONNET[3]; Amy, FELTY[2]; Daniel, GAFFÉ[4]; Annie, RESSOUCHE[3]; Franck, GRAMMONT[5]

1 Université Côte d'Azur, CNRS, I3S, France
2 School of Electrical Engineering and Computer Science, University of Ottawa, Canada
3 Université Côte d'Azur, INRIA SAM, France
4 Université Côte d'Azur, CNRS, LEAT, France
5 Université Côte d'Azur, CNRS, LJAD, France

Abstract Some specific neuronal graphs are known for having biologically relevant structures and behaviors and we call them archetypes. These archetypes are supposed to be the basis of typical instances of neuronal information processing. In this paper we report and compare two key approaches to the formal modeling and verification of neuronal archetypes. The first one exploits a synchronous programming language dedicated to reactive systems, the second one relies on a theorem prover.

Keywords Neuronal Networks, Leaky Integrate and Fire Modeling, Synchronous Languages, Model Checking, Theorem Prover, Coq.

1 Introduction

In the last years, much attention has been directed towards the study of the structure and function of brain networks. This research is often grouped under the now well-known keyword of the (human) *connectome* project [1, 2, 3]. However, although the research in this field claims to do graph analysis of the connectome at micro and macro scales, it mainly focuses on macro scales and is mostly based on diffusion or functional MRI data in humans [4].

Here, we want to promote a more fundamental and formal approach to the study of specific neuronal microcircuits, namely *neuronal archetypes*. From the biological point of view, the theory of neuronal archetypes postulates that the most primitive circuits that emerge during phylogenetic and ontogenetic evolution (i.e., in the first living systems and mostly in the mammal spinal cord and brain stem, or homolog in other phyla) are elementary circuits of a few neurons fulfilling a specific computational function (e.g., contralateral inhibition; see Section 4). The idea is that archetypes constitute the normalized form of potentially bigger and topologically more complicated neuronal circuits, but not more complex, in the sense that they do not perform computations other than the reference archetype itself does. In other words, every micro-circuit, even with many neurons, can theoretically be reduced to one of the few existing archetypes. Archetypes can be coupled in different ways. If the resulting circuit does not perform a more complex function in terms of neuronal information processing, it means that it should theoretically be reducible to a smaller archetype. If a new specific function, biologically relevant, is identified, then it can be categorized as a new and bigger archetype. From the informational point of view, neuronal archetypes would thus constitute the words of a finite dictionary. Following this analogy, neurons would be the letters that form the syllables constituting words. Some words can be coupled, thus acquiring a more or less different meaning, and all the words can be concatenated to build (meaningful) sentences.

As an example of well-known archetype, locomotive motion and other rhythmic behaviors are controlled by specific neuronal circuits called Central Generator Patterns (CPG) [5]. These CPGs have the capacity to generate oscillatory activities, at various regimes (under various different conditions), thanks to some specific properties at the circuit level.

It is relevant to investigate the dynamic behavior of all the possible archetypes of 2, 3 or more neurons, up to considering archetypes of archetypes. The aim is to see whether the properties of these archetypes of archetypes

simply are an addition of the individual constituent archetypes properties or something more. Since it would not be feasible to prove the properties of archetypes expected from the biological theory through real biological experiments, we exploit formal methods, and this is our originality. Formal methods thus help us is answering to some questions related to theoretical Neurosciences. In this paper, we report and compare the first attempts in the literature to apply *formal methods* of computer science to model and verify the temporal properties of fundamental neuronal archetypes in terms of neuronal information processing. We focus on the behavior of different basic archetypes and, for each one of them, we propose one or two representative properties that have been identified after extensive discussions with neurophysiologists [6, 7, 8]. From an electronic perspective, we consider archetypes as biologically inspired logical operators, which are easily adjustable by playing with very few parameters.

To model neuronal archetypes, we focus on Boolean Spiking Neural Networks where the neurons electrical properties are described via the *leaky integrate and fire* (LI&F) model [9]. Notice that discrete modeling is well suited to this kind of model because neuronal activity, as with any recorded physical event, is only known through discrete recording (the recording sampling rate is usually set at a significantly higher resolution than the one of the recorded system, so that there is no loss of information). We describe neuronal archetypes as weighted directed graphs whose nodes represent neurons and whose edges stand for synaptic connections. At each time unit, all the neurons compute their membrane potential, accounting not only for the current input signals, but also for the ones received along a given temporal window. Each neuron can emit a spike when a given threshold is exceeded. This kind of modeling is more sophisticated than the one proposed by McCulloch and Pitts in [10], where the behavior of a neural network is expressed in terms of propositional logic and the present activity of each neuron does not depend on past events.

Formal verification was initially introduced to prove that a piece of software or hardware is free of errors [11] with respect to a given model. The system at issue is generally modeled as a transition graph where each node represents a state of the system and each edge stands for a transition from a source to a destination state. Model checkers or theorem provers are often used to verify that specific properties of the system hold at particular states. The field of systems biology is a more recent application area for formal verification, and such techniques have turned out to be very useful so far in this domain [12]. A variety of biological systems can be modeled as graphs whose nodes represent the different possible configurations of the system and whose edges encode meaningful configuration changes. It is then possible to define and prove properties concerning the temporal evolution of the biological species involved in the system [13, 14]. This often allows deep insight into the biological system at issue, in particular concerning the biological transitions governing it, and the reactions the system will have when confronted with external factors such as disease, medicine, and environmental changes [15, 16].

The first formal approach we focus on, which was introduced by a subset of the authors of this paper [6, 7], is based on the modeling of neural networks using a *synchronous language* for the description of reactive systems (Lustre). Spiking neural networks can indeed be considered as reactive systems: their inputs are physiological signals coming from input synapses, and their outputs represent the signals emitted in reaction. This class of systems fits well with the synchronous approach based on the notion of logical time [17]: time is considered as a sequence of logical discrete instants. An instant is a point in time where external input events can be observed, along with the internal events that are a consequence of the latter. Inputs and resulting outputs all occur simultaneously. The synchronous paradigm is now well established relying on a rigorous semantics and on tools for simulation and verification.

Several synchronous languages respect this synchronous paradigm, and all of them have a similar expressivity. We choose Lustre [17], which defines operator networks interconnected with data flows and it is particularly well suited to expressing neuron networks. It is a data flow language offering two main advantages: (1) it is *functional* with no complex side effects, making it well adapted to formal verification and safe program transformation; (2) it is a *parallel* model, where any sequencing and synchronization depends on data dependencies. Moreover, the Lustre formalism is close to temporal logic and this allows the language to be used for both writing programs and expressing properties as observers [18]. An observer of a property is a program, taking as inputs the inputs/outputs of the model under verification, and deciding at each instant whether the property is violated or not. The tools automatically checking the property satisfaction/violation are called *model checkers*. There exist several model checkers for Lustre that are well suited to our purpose: *Lesar* [19], *Nbac* [20], *Luke* [21], *Rantanplan* [22] and *kind2* [23, 24]. After comparing all these model checkers [6], the most powerful one turned out to be kind2, which relies on SMT (Satisfiabitily Modulo Theories) based k-induction and combines several resolution engines.

The second approach we report on, which was also introduced by a subset of the authors of this paper [8], is a theorem proving approach. It is based on the use of the *Coq Proof Assistant* [25] to prove important properties of neurons and archetypes. Coq implements a highly expressive higher-order logic in which we can directly introduce datatypes modeling neurons and archetypes, and express properties about them. As a matter of fact, one

of the main advantages of using Coq is the generality of its proofs. Using such a system, we can prove properties about arbitrary values of parameters, such as any length of time, any input sequence, or any number of neurons. We use Coq's general facilities for structural induction and case analysis, as well as Coq's standard libraries that help in reasoning about rational numbers and functions on them. Our development does not depend on advanced features of Coq like dependent types or the hierarchy of universes, and thus can likely be translated into other theorem provers like the PVS Specification and Verification System [26] and Isabelle [27] quite easily.

To the best of our knowledge, the two approaches reported in this paper (the model-checking oriented approach and the theorem proving approach) are the only approaches to the formal verification of neuronal archetypes. The paper is organized as follows. In Section 2, we present the state of the art concerning neural network modeling and, more generally, modeling and formal verification of biological systems. In Section 3, we present a discrete version of the LI&F model. Section 4 is devoted to the description of the neuronal archetypes we take into consideration. In Section 5, we briefly introduce the Lustre Language and model checking techniques. In Section 6, we show archetype behaviors (encoded in Lustre) and in Section 7, we tackle the next logical step and deal with some archetype couplings. In Section 8, we introduce the Coq Proof Assistant. In Section 9, we present the Coq model for neural networks, which includes definitions of neurons, operations on them, and combining them into archetypes. In Section 10, we present and discuss the Coq specification and proof of three representative properties (the first two properties are intentionally two of the properties of Section 6). Finally, in Section 11, we summarize and compare the two proposed approaches on the formal verification of neuronal archetypes and give some future research directions. This paper has the value of rigorously comparing the approaches proposed in the conference papers [6, 7, 8]. Furthermore, it extends these papers by introducing new archetype definitions and properties. The first author of this paper belongs to the intersection of the author lists of the cited papers.

2 Related Work

In the last decades, there has been much promising research in the field of formal modelling and verification of biological systems. As far as the modelling of biological systems is concerned, in the literature we can find both qualitative and quantitative approaches. To express the qualitative nature of dynamics, the most used formalisms are Thomas' discrete models [28], Petri nets [29], p-calculus [30], bio-ambients [31], and reaction rules [32]. To capture the dynamics from a quantitative point of view, ordinary or stochastic differential equations are often used. More recent approaches include hybrid Petri nets [33] and hybrid automata [34], stochastic p-calculus [35], and rule-based languages with continuous/stochastic dynamics such as Kappa [36]. Relevant properties concerning the obtained models are then often expressed using a formalism called temporal logic and verified thanks to model checkers such as NuSMV [37] or PRISM [38].

Concerning the theorem proving approach, in [39] the authors propose the use of modal linear logic as a unified framework to encode both biological systems and temporal properties of their dynamic behavior. They focus on a model of the P53/Mdm2 DNA-damage repair mechanism and they prove some desired properties using theorem proving techniques. In [40], the Coq Proof Assistant is exploited to prove two theorems linking the topology and the dynamics of gene regulatory networks. In [41], the authors advocate the use of higher-order logic to formalize reaction kinetics and exploit the HOL Light theorem prover to verify some reaction-based models of biological networks. Finally, the Porgy system is introduced in [42]. It is a visual environment which allows modeling of biochemical systems as rule-based models. Rewriting strategies are used to choose the rules to be applied.

As far as neuronal networks are concerned, their modeling is classified into three generations in the literature [43, 44]. First generation models, based on McCulloch-Pitts neurons [10] as computational units, handle discrete inputs and outputs. Their computational units consist of a set of logic gates with a threshold activation function. Second generation models, whose most representative example is the multi-layer perceptron [45], exploit real valued activation functions. These networks, whose real-valued outputs represent neuron firing rates, are widely used in the domain of artificial intelligence. Third generation networks, also called *spiking neural networks* [44], are characterized by the relevance of time aspects. Precise spike firing times are taken into account. Furthermore, they consider not only current input spikes but also past ones (temporal summation). In [46], spiking neural networks are classified with respect to their biophysical plausibility, that is, the number of behaviors (i.e., typical responses to an input pattern) they can reproduce. Among these models, the Hodgkin-Huxley model [47] is the one able to reproduce most behaviors. However, its simulation process is very expensive even for a few neurons and for a small amount of time. In this work, we choose to use the leaky integrate and fire (LI&F) model [48], a computationally efficient approximation of a single-compartment model, which proves to be amenable to formal verification.

In addition to the works reviewed in this paper [6, 7, 8], there are a few attempts at giving formal models for spiking neural networks in the literature. In [49], a mapping of spiking neural P systems into timed au-

tomata is proposed. In that work, the dynamics of neurons are expressed in terms of evolution rules and durations are given in terms of the number of rules applied. Timed automata are also exploited in [50] to model LI&F networks. This modeling is substantially different from the one proposed in [49] because an explicit notion of duration of activities is given. Such a model is formally validated against some crucial properties defined as temporal logic formulas and is then exploited to find an assignment for the synaptic weights of neural networks so that they can reproduce a given behavior.

3 Discrete Leaky Integrate and Fire Model

In this section, we introduce a discrete (Boolean) version of LI&F modeling. We first present the basic biological knowledge associated to the modeled phenomena and then we detail the adopted model.

When a neuron receives a signal at one of its synaptic connections, it produces an excitatory or an inhibitory *post-synaptic potential* (PSP) caused by the opening of selective ion channels according to the post-synaptic receptor nature. An inflow of cations in the cell leads to an activation; an inflow of anions in the cell corresponds to an inhibition. This local ions flow modify the membrane potential either through a depolarization (excitation) or a hyperpolarization (inhibition). Such variations of the membrane potential are progressively transmitted to the rest of the cell. The potential difference is called *membrane potential*. In general, several to many excitations are necessary for the membrane potential of the post-synaptic neuron to exceed its *depolarization threshold*, and thus to emit an *action potential* at its axon hillock to transmit the signal to other neurons.

Two phenomena allow the cell to exceed its depolarization threshold: the *spatial summation* and the *temporal summation* [51]. Spatial summation allows to sum the PSPs produced at different areas of the membrane. Temporal summation allows to sum the PSPs produced during a finite time window. This summation can be done thanks to a property of the membrane that behaves like a capacitor and can locally store some electrical loads (*capacitive property*).

The neuron membrane, due to the presence of leakage channels, is not a perfect conductor and capacitor and can be compared to a resistor inside an electrical circuit. Thus, the range of the PSPs decreases with time and space (*resistivity* of the membrane).

A LI&F neuron network is represented with a weighted directed graph where each node stands for a neuron soma and each edge stands for a synaptic connection between two neurons. The associated weight for each edge is an indicator of the weight of the connection on the receiving neuron: a positive (resp. negative) weight is an activation (resp. inhibition).

The depolarization threshold of each neuron is modeled via the *firing threshold* τ, which is a numerical value that the neuron membrane potential p shall exceed at a given time t to emit an action potential, or *spike*, at the time $t+1$.

The membrane resistivity is symbolized with a numerical coefficient called the *leak factor* r, which allows to decrease the range of a PSP over time.

Spatial summation is implicitly taken into account. In our model, a neuron u is connected to another neuron v via a single synaptic connection of weight w_{uv}. This connection represents the entirety of the shared connections between u and v. Spatial summation is also more explicitly taken into account with the fact that, at each instant, the neuron sums each signal received from each input neuron. The temporal summation is done through a sliding integration window of length σ for each neuron to sum all PSPs. Older PSPs are decreased by the leak factor r. This way, the biological properties of the neuron are respected and the computational load remains limited. This allows us to obtain finite state sets, and thus to easily apply model checking techniques.

More formally, the following definition can be given:

Definition 1 ***Boolean Spiking Integrate and Fire Neural Network.*** A spiking Boolean integrate and fire neural network *is a tuple* (V, E, w)*, where:*

- V *are Boolean spiking integrate and fire neurons,*
- $E \subseteq V \times V$ *are synapses,*
- $w : E \to \mathbb{Q} \cap [-1, 1]$ *is the synapse weight function associating to each synapse* (u, v) *a weight* w_{uv}*.*

A spiking Boolean integrate and fire neuron *is a tuple* (τ, r, p, y)*, where:*

- $\tau \in \mathbb{Q}^+$ *is the* firing threshold*,*
- $r \in \mathbb{Q} \cap [0, 1]$ *is the* leak factor*,*
- $p : \mathbb{N} \to \mathbb{Q}$ *is the [membrane]* potential *function defined as*

$$p(t) = \begin{cases} \sum_{i=1}^{m} w_i \cdot x_i(t), & if\ p(t-1) \geqslant \tau \\ \sum_{i=1}^{m} w_i \cdot x_i(t) + r \cdot p(t-1), & otherwise \end{cases}$$

where $p(0) = 0$*,* m *is the number of inputs of the neuron,* w_i *is the weight of the synapse connecting the* i^{th} *input neuron to the current neuron, and* $x_i(t) \in \{0, 1\}$ *is the signal received at the time* t *by the neuron through its* i^{th} *input synapse (observe that, after the potential exceeds its threshold, it is reset to* 0*),*

- $y : \mathbb{N} \to \{0, 1\}$ *is the neuron output function, defined as*

$$y(t) = \begin{cases} 1 & if\ p(t) \geqslant \tau \\ 0 & otherwise. \end{cases}$$

(for the Lustre implementation, we set $y = 1$ *if* $p(t-1) \geqslant \tau$ *in order to prevent neurons from receiving and emitting signals at the same time unit.)*

The development of the recursive equation for the membrane potential function and the introduction of a sliding time window of length σ lead to the following equation for p (when $p(t-1) < \tau$): $p(t) = \sum_{e=0}^{\sigma} r^e \sum_{i=1}^{m} w_i \cdot x_i(t-e)$, where e represents the time elapsed until the current one.

4 The Basic Archetypes

The six basic archetypes we study are the following ones (see Fig. 1). These archetypes can be coupled to potentially constitute a bigger one.

- **Simple series** is a sequence of neurons where each element of the chain receives as input the output of the preceding one.
- **Series with multiple outputs** is a series where, at each time unit, we are interested in knowing the outputs of all the neurons (i.e., all the neurons are considered as output ones).
- **Parallel composition** is a set of neurons receiving as input the output of a given neuron.
- **Negative loop** is a loop consisting of two neurons: the first neuron activates the second one while the latter inhibits the former.
- **Inhibition of a behavior** consists of two neurons, the first one inhibiting the second one.
- **Contralateral inhibition** consists of two or more neurons, each one inhibiting the other ones.

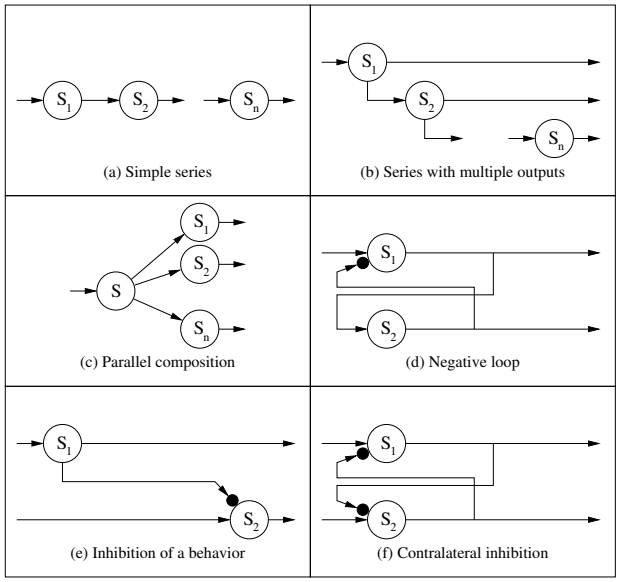


Fig. 1 The basic neuronal archetypes.

5 The synchronous language Lustre and Model Checking

As explained in Section 1, we adopt the synchronous language Lustre [17] to describe neuronal behaviors with a declarative modeling approach. Such a programming language is based on the *synchronicity hypothesis*, which is characterized by the concept of *logical time*. Lustre considers *time* as a sorted discrete *flow* of signals. At each time unit (*clock* value), systems react to inputs and generate outputs at the same instant. A Lustre code displays time dependencies between signals and explains them thanks to a set of equations. This mechanism allows to express time as an infinite sequence of (natural) values. A Lustre program behaves as a cyclic system: all the present variables take their nth value at the nth execution step.

The basic structure of a Lustre program is the *node*. In a Lustre node, equations, expressions, and assertions are used to express output variable sequences of values from input variable sequences. Variables are typed: types can be either basic (Boolean, integer, real), or complex (structure or vector). Complex types are defined by the user. Usual operators over basic types exist: `+, -, ...`; `and, or, not`; `if then else`. These data operators only work with variables sharing the same clock. The result shares the same clock too. Furthermore, Lustre disposes of two main temporal operators to handle logical time represented by clocks. These operators are `pre` and $\rightarrow$:

- `pre` (for previous) reacts as a memory: if $(e_1, e_2, ..., e_n, \dots)$ is the flow `E`, `pre(E)` is the flow $(nil, e_1, e_2, ..., e_n, \dots)$, where *nil* is the undefined value denoting uninitialized memory.
- $\rightarrow$ (meaning "followed by") is used with the `pre` operator and prevents from having uninitialized memory: let `E` $= (e_1, e_2, \dots, e_n, \dots)$ and `F` $= (f_1, f_2, \dots, f_n, \dots)$ be two flows, then `E`$\rightarrow$`F` is the expression $(e_1, f_2, \dots, f_n, \dots)$.

Equations define output variables in nodes. Moreover, assertions can be exploited to force variable values. Assertions allow to take the environment into account by making assumptions which consist in boolean expressions. These expressions must always be true. For instance, the assertion:
`assert (true` $\rightarrow$ `(not x) or (pre(not x)))` says that in the value sequence of x, no two consecutive `true` exist.

Lustre is a unifying framework: one the one side, it allows to model reactive systems, on the other side it gives the possibility to express some temporal properties concerning the modeled systems. These properties are written as Lustre nodes called *observers* [18], which verify the outputs of the program under verification for each possible instant, and return `true` if the encoded behavior is satisfied, `false` otherwise. The tool kind2 [23, 24]

allows to make automatic verifications. We have chosen it for its compatibility with the Lustre syntax and for its efficiency compared to other model checkers such as Lesar [19] and Nbac [20]. If the given property is not true for all possible input variable values, then kind2 gives a counterexample, which consists in an execution trace leading to the violation of the property.

6 Encoding Neuronal Archetypes and Temporal Properties in Lustre

Lustre allows to easily model and encode neuron behaviors. An input matrix (`mem`) is used to record present and past received signals. For each instant, the leftmost column of this matrix stores the (weighted) current inputs and, for the other columns, values are defined as follows: (i) they are equal to 0 at the first instant (initialization) and (ii) for all the next instants, they are reset to 0 in case of spike emission at the previous instant and they take the preceding time unit value of their own left column (for the corresponding row) otherwise. This process implements a sliding time window and is encoded with the `pre` operator. Such an input matrix is multiplied, at each instant, by a vector of remaining coefficients (`rvector`) and, whenever the firing threshold is reached, a spike is emitted (at the next time unit). The code defining a Lustre neuron with one input data stream is given below:

```
node neuron (X: bool; w:int) returns(Spike: bool);
var threshold, V: int;
    rvector: int^5;
    mem: int^5;
    localS: bool;
let
   threshold=105;
   rvector[0..4]=[10,5,3,2,1];
   V=mem[0]*rvector[0]+mem[1]*rvector[1]+
     mem[2]*rvector[2]+mem[3]*rvector[3]+
     mem[4]*rvector[4];
   localS=(V>=threshold);
   mem[0]=if X then w else 0;
   mem[1..4]=[0,0,0,0]->if pre(localS) then 0
                        else pre(mem[0..3]);
   Spike= false -> pre(localS);
   -- reaction at the next instant
tel
```

All the values are multiplied by ten in order to work in fixed point precision, and thus have better model checking performances. Notice that all the constants of this Lustre node can be given as parameters as follows:

```
node neuron (X: bool; w, threshold: int;
             rvector: int^5)
 returns(Spike: bool);
```

Lustre is a modular language and, thanks to this feature, archetypes can be directly encoded from basic neurons.

In the rest of the section, we focus on relevant properties of neurons and archetypes (we use 1 to denote `true` and 0 to denote `false`).

6.1 Single Neuron

Firstly, we concentrate on single neuron behaviors. Whatever the parameters of a neuron are (input synaptic weights, firing threshold, leak factor, length of the integration window), it can only behave in one of the two following ways: (i) the sum of the current entries (multiplied by their weights) allows to reach the threshold (and thus, for each time units when the neuron gets enough input signals, it emits a spike at the next instant), or (ii) several time units are required to overtake the threshold. More formally, given a neuron with a unique input, property 1 has been checked thanks to kind2:

Property 1 ***[Single neuron.]*** *Given a neuron receiving an input flow on the alphabet* $\{0, 1\}$*, it can only express one of the two exclusive following behaviors:*

Delayer effect. *It emits a* 0 *followed by a flow identical to the input.*

Filter effect. *It emits at least two occurrences of* 0 *at the beginning and can never emit two consecutive occurrences of* 1*.*

From a biological point of view, in the first case (delayer) we can speak of instantaneous integrator and in the second case (filter) of long time integrator. In the case of a delayer, the neuron just emits a sequence identical to the input one, with a delay of one time unit. In the case of a filter, the neuron only transmits one signal out of x ($1/x$ filter). Note that if the neuron is not able to overtake its threshold (*wall* effect), we have a limit case of the filter effect. The Lustre observer of property 1, consisting of an exclusive conjunction between a delayer and a filter, is given in the code below.

```
node delayer (X: bool; w: int) returns(OK: bool);
  var
    SX, Out, S1, verif, preverif: bool;
  let
    Out = neuron(X, w);
    S1 = true -> Out;
    SX = false -> pre(X);
    verif  = true->if SX then S1 else false;
    preverif = true->pre(verif);
    OK = if preverif and verif then true else false;
  tel


node filter (X: bool; w: int) returns(OK: bool);
  var
    Out: bool;
  let
    Out = neuron(X,w);
    OK =true -> if Out then not pre(Out)
                else not Out;
  tel
```

```
node prop1 (X: bool; w: int) returns (OK: bool);
  var
    S1,S2, Verif: bool;
  let
    S1 = delayer(X, w);
    S2 = filter(X, w);
    Verif = S1 XOR S2;
    OK = confirm_property_2_ticks(X) or Verif;
  tel
```

6.2 Simple Series (see Fig. 1(a))

A simple series of length n behaves according to the neurons composing it. There are two cases: (i) the series contains n delayers (and in this case it acts as a delayer of n time units), or (ii) it contains at least one filter (it thus shows a n-delayer effect composed of a filter effect). More formally, the following property can be given:

Property 2 ***[n-delayer or n-delayer/filter.]*** *Given a series of length n receiving an input flow on the alphabet* $\{0, 1\}$*, it can only express one of the two exclusive following behaviors:*

n-delayer effect. *It emits a sequence of* 0 *of length n followed by a flow identical to the input one.*

n-delayer/filter effect. *It emits a sequence of* 0 *of length at least* $n + 1$ *and can never emit two consecutive* 1*.*

A consequence of property 2 is that a simple series cannot constitute a permanent signal, e.g., if it receives an oscillatory signal as input, it is not able to emit a sequence of 1 as output. We can also point out that filter neurons do *not commute* in a simple series. For instance, if in a series of two neurons a 1/2 filter (that is, a neuron emitting a 1 every two instants when receiving a sequence of 1) precedes a 1/3 filter, the result is a 1/6 filter. If the two neurons are inverted, the result is a wall effect (no signals are emitted). In order to avoid wall effects, the most selective neurons should thus be the first ones.

6.3 Series with Multiple Outputs (see Fig. 1(b))

In a series with multiple outputs, the emission of each neuron depends by the emissions of the preceding neurons. Furthermore, the following property is valid:

Property 3 ***[Exclusive temporal activation in a series with multiples outputs.]*** *When a series of n delayers with multiples outputs receives the output of a* $1/n$ *filter, only one neuron at a time overtakes its threshold (and thus emits a spike).*

As a consequence, in the configuration of Property 3 two neurons can not emit at the same time.

6.4 Parallel Composition (see Fig. 1(c))

As far as the parallel composition is concerned, the number of spikes emitted in parallel at each instant is in between a given interval, whose upper bound is not necessarily the number of neurons in parallel (because, even if each single neuron has the capability to emit, the parallel neurons can be unsynchronized, that is, not able to emit simultaneously).

Moreover, the following relevant property holds:

Property 4 ***[Parallel composition of n filters.]*** *Given a parallel composition with a delayer connected to n filters of different selectivity connected to the same delayer, it is possible to emit as output a sequence of* 1 *of length k, with* $k \geq n$*.*

To have a sequence of 1 of length k, the key idea is to take the parallel composition of n filters of different selectivity $1/X_i$, where $i \in \{1, ., n\}$ and X_1,...,X_n is the set of prime numbers in between 2 and k. Moreover, the parallel composition can be exploited to constitute a permanent signal from a not permanent one.

6.5 Negative Loop (see Fig. 1(d))

If, without considering the inhibiting edge, the two neurons of a negative loop operate as delayers, then the two neurons show an oscillatory output behavior (provided that a permanent signal is injected in the archetype). More precisely, the following property is verified:

Property 5 ***[Oscillation in a negative loop.]*** *Given a negative loop composed of two delayers, when a sequence of* 1 *is given as input, the activator neuron oscillates with a pattern of the form* 1100 *(and the inhibitor expresses the same behavior delayed of one time unit).*

6.6 Inhibition of a Behavior (see Fig. 1(e))

For a large class of neuron parameters (that is, firing threshold, leak factor, and length of the integration window), property 6 holds:

Property 6 ***[Fixed point inhibition.]*** *Given an inhibition archetype, if a sequence of* 1 *is given as input, at a certain time the inhibited neuron can only emit* 0 *values.*

6.7 Contralateral Inhibition (see Fig. 1(f))

It has been shown that there is a set of neuron parameters for which the following behavior can be observed.

Property 7 ***[Winner takes all in a contralateral inhibition.]*** *Given a contralateral inhibition archetype with two or more neurons, if a sequence of* 1 *is given as input, starting from a given time one neuron is activated and the other ones are inhibited.*

The intuition is that the most excited neuron becomes the most inhibitory one, and even if it is itself inhibited, it necessarily is less than its neighbors.

Some additional properties related to the different archetypes can be found in [52]. We are currently studying some supplementary archetypes such as the positive loop, where a first neuron activates a second neuron, which in turn activates the former one. While the negative loop is very frequent and allows to regulate systems, the negative loop is less frequent. On one hand, it allows to extend signals in time, on the other hand it can provoke a chain reaction where the system activates itself up to reaching its maximum, without having the capability of leaving it. It is interesting to find the conditions under which this phenomenon can be observed.

7 Archetype Coupling in Lustre

Two archetypes can be coupled in the following ways: (i) by connecting the output(s) of one archetype to the input(s) of the other one, that is, by making a concatenation, (ii) by nesting one archetype within the other one. In this section we introduce some representative couplings among the ones we treated. In the following figures, these acronyms are used: A for activator, I for inhibitor, D for delayer, F for filter, G for pattern generator, S for series, and C for collector.

7.8 Simple Series within Negative Loop

We start considering a series of n delayers nested between the activator and the inhibitor of a negative loop, as illustrated in Figure 2. We remind that, when a sequence of 1 is injected in the negative loop archetype, an oscillating output of the form 1100 is produced (see Property 5). The addition of the series involves an augmentation of the oscillation period, which passes from 2 to n. More precisely, the following property holds:

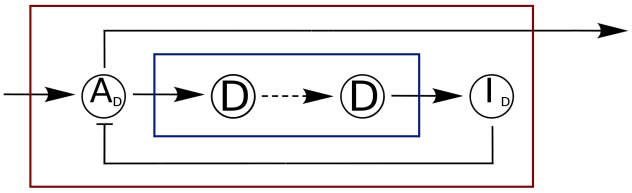


Fig. 2 Series of n delayers nested between the activator and the inhibitor of a negative loop.

Property 8 ***[Oscillation period extension.]*** *Given a simple series of delayers of length n within a negative loop, if a sequence of* 1 *is given as input, the output of the activator is of the form :* $0(1^{n+2}0^{n+2})^{\omega}$ *(we recall that x^y denotes the repetition of x for y time units and x^{ω} denotes the infinite repetition of x.)*

7.9 Concatenation of Simple Series and Negative Loop

We consider now a series of n delayers preceding a negative loop, as illustrated in Figure 3. In this case, the oscillation period is not modified but the beginning of the oscillation is delayed. More in detail:

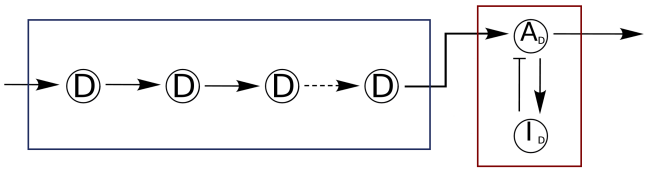


Fig. 3 Series of n delayers connected to the activator of a negative loop.

Property 9 ***[Oscillation delay.]*** *Given a simple series of delayers of length n connected to the activator of a negative loop, if a sequence of* 1 *is given as input, the output of the activator is of the form :* $0^n(1100)^{\omega}$.

As a next step, it is important to identify all the input patterns of the negative loop producing an oscillatory trend output. Since the simple series is only able to transmit or filter signals (see Property 2), in the next subsection we present a neuron combination which is able to produce all the patterns of a fixed length on the alphabet $\{0, 1\}$.

7.10 Concatenation of Periodic Pattern Generator and Negative Loop

Let us consider the system which is graphically depicted in Figure 4. As input of the system, a sequence of 1 is received by a $1/n$ filter connected to a series of n delayers with multiple outputs, which is then connected to a collector delayer neuron. One can choose to activate or deactivate each one of the edges linking the neurons of the series to the collector. We could prove that this system can generate all the patterns of length n on the alphabet $\{0, 1\}$. To get the different patterns, the intuition is to play with the activation/deactivation combinations of the edges connecting the neurons of the series to the collector neuron.

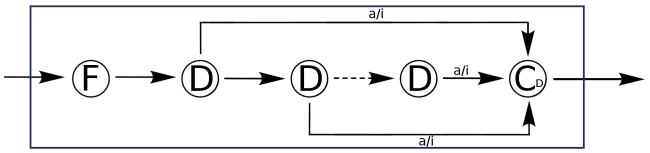


Fig. 4 Generator of periodic patterns based on a series with multiple outputs.

Another example of pattern generator is illustrated in Figure 5.

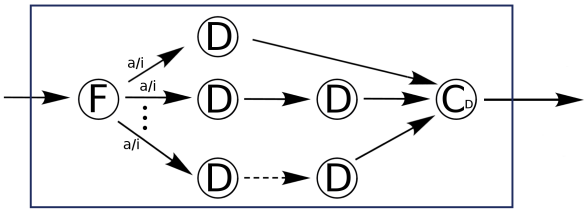


Fig. 5 Generator of periodic patterns based on a parallel composition.

In this system, a $1/n$ filter is connected to n simple series of delayers, of increasing length from 1 to n. The edges linking the filter to the series can be activated or deactivated (the key idea is that the activation of the series of length x allows the emission of a 1 in the x-position of the pattern). While the first generator has a number of nodes and edges which is linear with respect to the length of the pattern to generate, in the second generator the number of nodes and edges is quadratic. For this reason, we employ the first generator for our studies.

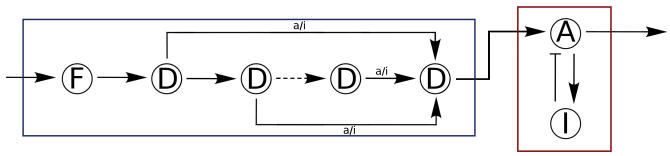


Fig. 6 Generator of periodic patterns connected to the activator of a negative loop.

For different values of n, we connected the pattern generator to the input of the negative loop (see Figure 6) and we retrieved all the patterns able to produce oscillations. We take into account not only strict Square Oscillations, where the number of 1 equals the number of 0, but also Pulse Wave Modulations with a ratio of almost fifty percent (where the difference between the consecutive number of 1 and the consecutive number of 0 is at most 2).

For instance, for $n = 5$, the pattern 11001 generates oscillations of the form 11000 as output of the negative loop (we recall that a simple series getting a sequence of 1 as input cannot emit a pattern of the form 11001).

7.11 Series within Contralateral Inhibition

In this subsection we study the integration of a simple series of n delayers within a two neurons contralateral inhibition to defer the inhibition of the losing neuron. At this purpose, we use a first neuron N_1 acting as a delayer (if inhibitions are neglected) connected to a simple series S_n of n delayers, which inhibits a second neuron N_2. The second neuron (which is a delayer too) inhibits N_1 and the second inhibition is weaker than the first one (see Figure 7).

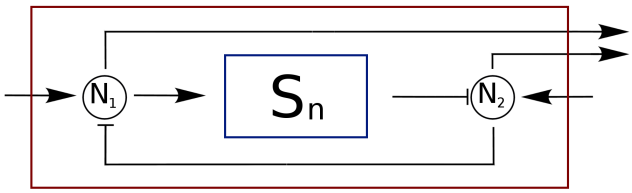


Fig. 7 Series of n delayers within a contralateral inhibition of two neurons.

The goal is to understand how the winner takes all behavior (see Property 7) is affected when the inhibition of the loser neuron is deferred. We verify that, for several inhibitor edge weights, the delayer series introduction makes the system stabilize later. Furthermore, as shown Figure 8, the stabilization is preceded by $n + 1$ damped oscillations of period $n + 2$.

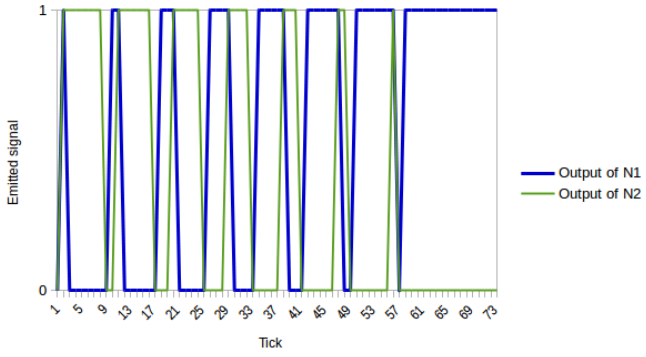


Fig. 8 Output of N1 and N2 for a simple series of $n = 6$ delayers nested in a two neurons contralateral inhibition (see Figure 7).

The first oscillation of N_1 consists of one 1 and a number of 0 equal to $n + 1$. For each subsequent oscillation, the number of 1 (resp. of 0) increases (resp. decreases) of one unit. After its last oscillation, N_1 emits an infinite

sequence of 1. The behavior of N_2 is symmetric (its first oscillation is composed of one 0 and $n+1$ occurrences of 1).

The following property has been modeled as a Lustre observer and proved for multiple inhibitor edge weights:

Property 10 ***[Winner takes all delay.]*** *Let us consider a delayer neuron N_1 connected to a series of n delayers S_n inhibiting a delayer neuron N_2, which in turn inhibits N_1. The edge inhibiting N_2 has a higher absolute value than the one inhibiting N_1. Let us suppose a sequence of 1 is given as input of the archetype composition. When N_1 emits a sequence of 1 as long as the first sequence of 1 emitted by N_2, then, after the emission of a 0 by N_1, $N1$ (resp. N_2) only emits a sequence of 1 (resp. 0).*

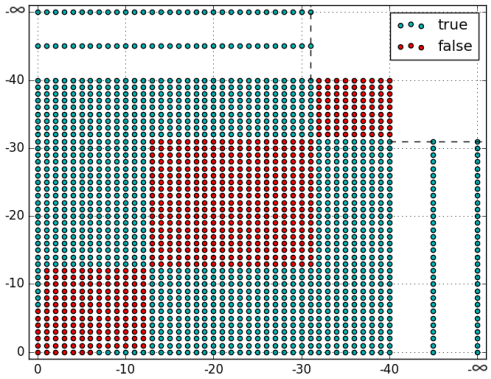


Fig. 9 Verification of the winner takes all behavior for the different values of the inhibiting weights of the two neurons in a simple contralateral inhibition.

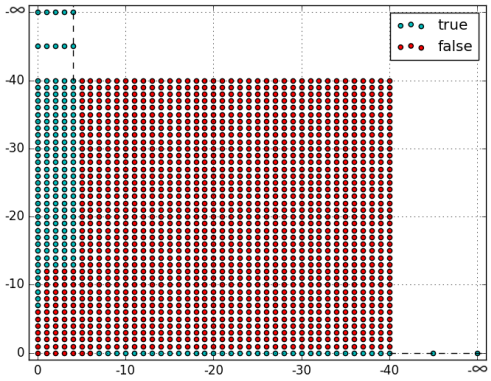


Fig. 10 Verification of the winner takes all behavior for the different values of the inhibiting weights of the two neurons in a contralateral inhibition with two neurons and the insertion of one single delayer.

The two diagrams of Figure 9 and Figure 10 allow to make the behavioral comparison between a two neuron contralateral inhibition and the same archetype equipped with a single delayer. In both plots, the y-axis (resp. x-axis) represents the weight of the edge inhibiting the first (resp. second) neuron. Blue (resp. red) points represent pairs of weight values for which the stabilization is (resp. is not) reached within the first four time units. We can identify that, passing from the first (Figure 9) to the second diagram (Figure 10), the red zone (non satisfaction of the winner takes all property) increases. Moreover, the growth of the red zone is asymmetric, which reflects the asymmetry of the archetype composition. Contrary to what is expected, the neuron proved to win more often within the first four time units is the one preceding the delayer series (even if its output inhibitor signal is delayed of one time unit).

Notice that the current observers we propose deal with infinite values but cannot cover the whole parameter space. In the future, we intend to define more subtle observers to improve the space covering. For instance, we plan to prove the stability (or the linear growth?) of the red regions of Figure 9.

7.12 Concatenation of Pattern Generator and Inhibition

As last representative example, we propose to concatenate the pattern generator with the inhibition archetype (see Figure 11).

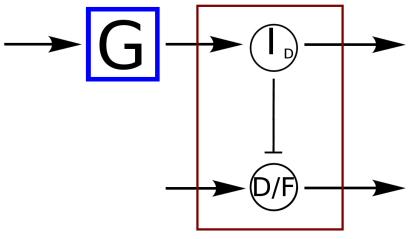


Fig. 11 Pattern generator followed by inhibition

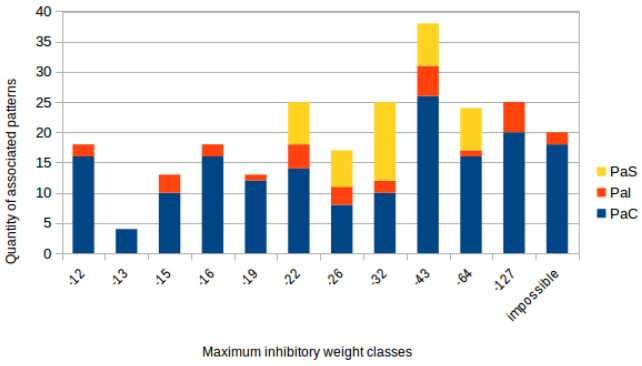


Fig. 12 Classes of maximal inhibitory weights allowing the inhibition behavior. The orange color corresponds to palindrome patterns (Pal), the blue color corresponds to patterns whose twin is in the same class (PaC), and the yellow color stands for patterns whose twin falls in another class (PaS).

As a first step, we kept the neuron parameters and edge weights obtained for the inhibition archetype alone, and searched for the input patterns able to entail the desired behavior (that is, starting from a given time, the inhibited neuron stops emitting spikes, see Property 6). With the previous parameters, the permanent sequence of 1 is the only pattern allowing to get the given behavior. We then succeeded in finding other patterns giving the inhibition property when the weight of the inhibiting edge is strengthened. We checked the property for every pattern of length $n < 8$. Patterns are classified with respect to the maximal inhibitor edge weight allowing to obtain the given behavior (histogram of Figure 12). We can see that, if a given pattern is not a palindrome (that is, its reading from the left to the right and from the right to the left is different), it necessarily has a *twin pattern* (which is obtained by scanning the given pattern from the right to the left, and is different from the given pattern). There are several cases in which a pattern and its twin are not in the same weight class. For instance, for $n = 7$, the pattern 0110100 and his twin 0010110 are in two different (neighbor) classes. We can deduce that in a pattern, the number of zeros and ones is not the only important feature: the way zeros are interleaved by ones is relevant.

8 The Coq Proof Assistant

In this section, we introduce the main Coq features we exploit for neural network modeling. Additional information on Coq can be found in [25, 53]. The full implementation of our model along with the properties and their proofs described in this section, the next section, and the appendix are available at

`http://www.site.uottawa.ca/~afelty/coq-archetypes/`.

Coq is a formal proof management system that implements the Calculus of Inductive Constructions [54], which is an expressive higher-order logic [55]. Such a system allows users to formalize and prove properties in this logic. Expressions in the logic include a functional programming language. Such a language is typed (every Coq expression has a type). For instance, `X:nat` says that variable `X` takes its value in the domain of natural numbers. The types we employ in our model include `nat`, `Q`, `bool`, and `list` which denote natural numbers, rational numbers, booleans, and lists of elements respectively. These types are available in Coq's standard libraries. All the elements of a list must have the same type. For example, `L:list nat` expresses that `L` is a list of natural numbers. An empty list is denoted by `[]` or `nil` in Coq. Functions are the basic components of functional programming languages. The general form of a Coq function is given below.

`Definition`/`Fixpoint` Function_Name
(Input$_1$: Type of Input$_1$) ...
(Input$_n$: Type of Input$_n$): Output Type :=
  Body of the function.

The Coq keywords `Definition` and `Fixpoint` are used to define non-recursive and recursive functions, respectively. These keywords are followed by the function name. After the function name, there are the input arguments and their corresponding types. Inputs having the same type can be grouped. For instance, `(X Y Z: Q)` states all variables `X`, `Y`, and `Z` are rational numbers. Inputs are followed by a colon, which is followed by the output type of the function. Finally, there is a Coq expression representing the body of the function, followed by a dot.

In Coq, pattern matching is exploited to perform case analysis. This useful feature is used, for instance, in recursive functions, for discriminating between base cases and recursive cases. For example, it is employed to distinguish between empty and nonempty lists. A nonempty list consists of the first element of the list (the *head*), followed by a double colon, followed by the rest of the list (the *tail*). The tail of a list itself is a list and its elements have the same type as the head element. For instance, let L be the list `(5::2::9::nil)` containing three natural numbers. In Coq, the list L can also be written as `[5;2;9]`, where the head is `5` and the tail is `[2;9]`. Thus, non-empty lists in Coq often follow the general pattern `(h::t)`. In addition, there are two functions in Coq library called `hd` and `tl` that return the head and the tail of a list, respectively. For example, `(hd d l)` returns the head of the list `l`. Here, `d` is a default value returned if `l` is an empty list and thus does not have a head. Also, `(tl l)` returns the tail of the list `l` and returns `nil` if there is no tail.

Another Coq data type is natural numbers. A natural number is either `O` or the successor of another

natural number, written `(S n)`, where `n` is a natural number. For instance, `1` is represented as `(S 0)`, `2` as `(S (S 0))`, etc. In the code below, some patterns for lists and natural numbers are shown using Coq's `match...with...end` pattern matching construct.

```
match X with
| 0 ⇒ calculate something when X = 0
| S n ⇒ calculate something when X is successor of n
end

match L with
| [] ⇒ calculate something when L is an empty list
| h::t ⇒ calculate something when L has head h
          followed by tail t
end
```

In addition to the data types that are defined in Coq libraries, new data types can be introduced. One way to do so is using Coq's facility for defining records. Records can have different fields with different types. For instance, we can define a record with three fields `Fieldnat`, `FieldQ`, and `ListField`, which have types natural number, rational number, and list of natural numbers, respectively. Fields in Coq records can also represent constraints on other fields. For instance, field `CR` in the code below states that field `Fieldnat` must be greater than 7. The Coq syntax for the definition of the full record is shown in the code below.

```
Record Sample_Record := MakeSample {
  Fieldnat: nat;
  FieldQ: Q;
  ListField: list nat;
  CR: Fieldnat > 7 }.

S: Sample_Record
```

A record is a type like any other type, and so for example, variables can have the new record type. Variable `S` with type `Sample_Record` is an example. When a variable of this record type gets a value, all the constraints in the record have to be satisfied. For example, `Fieldnat` of `S` cannot be less than or equal to 7.

9 Encoding Neurons and Archetypes in Coq

We start illustrating our formalization of neural networks in Coq with the code below.

```
Record Neuron := MakeNeuron {
  Output:list nat;
  Weights:list Q;
  Leak_Factor:Q;
  Tau:Q;
  Current:Q;
  Output_Bin: Bin_List Output;
  LeakRange: Qle_bool 0 Leak_Factor = true /\
             Qle_bool Leak_Factor 1 = true;
  PosTau: Qlt_bool 0 Tau = true;
  WRange: WeightInRange Weights = true }.

Fixpoint potential (Weights: list Q)
  (Inputs: list nat): Q :=
  match Weights, Inputs with
  | nil, _ => 0
  | _, nil => 0
  | h1::t1, h2::t2 => if (beq_nat h2 0%nat)
                      then (potential t1 t2)
                      else (potential t1 t2) + h1
  end.
```

To define a neuron, we use the Coq record structure. This record consists of five fields with their corresponding types, and four fields representing constraints that the first five fields must satisfy according to the LI&F model defined in Section 3. A neuron output `(Output)` is represented as a list of natural numbers, with one entry for each time step. The weights linked to the inputs of the neuron `(Weights)` are stored in a list of rational numbers (one for each input in some fixed order). The leak factor `(Leak_Factor)`, the firing threshold `(Tau)`, and the most recent neuron membrane potential `(Current)` are rational numbers. As far as the four conditions are concerned, `PosTau` states that `Tau` must be positive (i.e., `Qlt_bool 0 Tau = true` is the Coq representation for $0 < \tau$, which encodes the condition $\tau \in \mathbb{Q}^+$ from Definition 1). `Qlt_bool` and other arithmetic operators can be found in Coq's rational number library. The other three conditions state, respectively, that `Output` contains only 0s and 1s (it is a binary list), `Leak_Factor` is between 0 and 1 inclusive, and each input weight is in the interval [-1, 1]. We do not provide the definitions of `Bin_List` and `WeightInRange` used in these statements. The reader can consult the accompanying Coq code for details.

For each neuron `N`, we write `(Output N)` to denote its first field, and similarly for the other fields. To define a new neuron with values `O, W, L, T,` and `C` with the suitable types, and proofs `P1,..., P4` of the four constraints, we write `(MakeNeuron O W L T C P1 P2 P3 P4)`.

The next definition in the above code computes the weighted sum of the inputs of a neuron, which is fundamental for the calculation of the potential function of a neuron (see Definition 1). In this recursive function, there are two arguments: `Weights`, which represents some number m of weights $w_1, \ldots, w_m$, and `Inputs`, which represents m inputs $x_1, \ldots, x_m$. The function returns an element of type `Q`. Its definition employs pattern matching on both inputs at the same time. In the body of the definition there are booleans, the `if` statement, and the equality operator on natural numbers `(beq_nat)`, all from Coq's standard libraries. Natural number constants, such as `0%nat` above, are given with their types to differentiate them from rational number constants, whose types are omitted. Although, the `potential` func-

tion is always called with two lists of the same length, Coq requires functions to be total; if two lists have different length, we return a "default" value of 0 in the base case. Furthermore, when this function is called, `Inputs`, which is the second argument of the function, must be a binary list (that is, it can only contain the natural numbers 0 and 1). Thus, when the head of the list `h2` is 0, we do not need to add anything to the final sum because anything multiplied by 0 is 0. In such a case, we just call the function recursively on the remaining weights and inputs `t1` and `t2`, respectively. On the other hand, if `h2` is 1, we add `h1`, the head of `Weights`, to the final sum, which is the recursive call on `t1` and `t2`. We need to implement the potential function in this way because `h1` and `h2` cannot be multiplied in Coq because they have different types. Recall that `h1` is a rational number and `h2` is a natural number.

The following code illustrates the `NextPotential` function, which computes $p(t)$ from Definition 1.

```
Definition NextPotential
 (N: Neuron) (Inputs: list nat): Q :=
 if (Qle_bool (Tau N) (Current N))
 then (potential (Weights N) Inputs)
 else (potential (Weights N) Inputs) +
      (Leak_Factor N) * (Current N).
```

Recall that `(Current N)` is the most recent potential value of the neuron, which is $p(t-1)$ in Definition 1. `(Qle_bool (Tau N) (Current N))` represents $\tau \leq p(t-1)$, and we use the potential function defined before to compute the weighted sum of the neuron inputs. Finally, the last line computes $r \cdot p(t-1)$.

The following code gives two important definitions.

```
Definition NextOutput
  (N: Neuron) (Inputs: list nat): nat :=
  if (Qle_bool (Tau N) (NextPotential N Inputs))
  then 1%nat
  else 0%nat

Definition NextNeuron
  (N: Neuron) (Inputs: list nat): Neuron :=
  MakeNeuron
    ((NextOutput N Inputs)::(Output N))
    (Weights N)
    (Leak_Factor N)
    (Tau N)
    (NextPotential N Inputs)
    (NextOutput_Bin_List N Inputs (Output_Bin N))
    (LeakRange N)
    (PosTau N)
    (WRange N).
```

The first definition calculates the next output of the neuron, which is $y(t)$ in Definition 1. Recall that `(NextPotential N Inputs)` computes $p(t)$. The expression `(Qle_bool (Tau N) (NextPotential N Inputs))` thus encodes the constraint $\tau \leq p(t)$.

In our model, the state of each neuron is represented by the `Output` and `Current` fields. The `Output` field of a neuron in the initial state is `[0%nat]`, which represents a list containing one 0. The `Current` field denotes the initial potential value, which is set to 0. A neuron changes its state by processing its inputs. After treating a list of n inputs, the `Output` field becomes a list of length $n+1$ containing 0's and 1's, and the `Current` field is set to the value of the potential after processing these n inputs. A state change occurs by applying the `NextNeuron` function reported in the above code to a neuron and a list of inputs. We represent a neuron at its later state by generating a new record with the new values for `Output` and `Current`, and other values directly copied over. We store the values in the `Output` field in reverse order, which helps making proofs by induction over lists easier in Coq. Thus, the most recent output of the neuron is at the head of the list. This can be seen in the above code, where the new value of the output is `((NextOutput N Inputs)::(Output N))`. The next output of the neuron is at the head, and is followed by the previous outputs. `(NextPotential N Inputs)` is the new value for `(Current N)`. Recall that `(Current N)` is the most recent value of the potential value of the neuron, or $p(t-1)$. So, for computing the next potential value of the neuron or $p(t)$, the `NextPotential` function must be called.

Proofs of the four constraints result from the new values for each field of the neuron. The first one requires a lemma `NextOutput_Bin_List` (statement omitted) proving that the new longer list is (still) a binary list. Proofs of the other three constraints are carried over exactly from the original neuron, since they concern some components of the neuron that remain unchanged.

The `ResetNeuron` function is employed to reinitialize a neuron to the initial state.

```
Definition ResetNeuron
  (N: Neuron): Neuron := MakeNeuron
  ([0%nat])
  (Weights N)
  (Leak_Factor N)
  (Tau N)
  (0)
  (Reset_Output)
  (LeakRange N)
  (PosTau N)
  (WRange N).
```

This function takes any `Neuron` as input, and returns a new one, where the `Output`, `Current`, and `Output_Bin` fields are reset, while the other fields are unchanged. The `Reset_Output` property is a simple lemma stating that `[0%nat]` satisfies the `Bin_List` property.

So far, we have presented the encoding of single neurons in isolation from other neurons. We next consider archetypes. In general, our approach is to encode the particular structure of each archetype as a Coq record.

Using a record for each archetype facilitates stating and proving properties about them. Recall that archetypes are functional structures of neural networks. Defining them in this abstract way helps us to present their basic functions. To illustrate this approach, we introduce now the encoding of two archetypes, the simple series in Figure 1(a) and the negative loop in Figure 1(d). As shown in Figure 1(a), a simple series consists of a list of single input neurons. The first neuron receives the input of the archetype and sends its output to the second neuron. Starting from the second neuron each neuron receives its input from the previous neuron and sends its output as the input of the next neuron in the series. The last neuron produces the output of the series. The `NeuronSeries` record defined below represents this structure in Coq.

```
Record NeuronSeries {Input: list nat} :=
MakeNeuronSeries
{
  NeuronList: list Neuron;
  NSOutput: list nat;
  AllSingle: forall (N:Neuron),
    In N NeuronList ->
    (beq_nat (length (Weights N)) 1%nat) = true;
  SeriesOutput: NSOutput =
  (SeriesNetworkOutput Input NeuronList);
}.
```

Records can have input parameters, similar to functions in Coq, and here the list of inputs to the simple series is `Input: list nat`. Thus `NeuronSeries` is actually a function from a list of natural numbers to a record. Curly brackets around input arguments is Coq notation for *implicit* arguments, which are arguments that can be omitted from expressions as long as Coq can figure out the missing information. Its use here allows us to write more readable Coq code. `NeuronList` is a field in the record representing the list of neurons in the simple series. The first element in this list is the first neuron is the series, etc. `NSOutput` represents the list of outputs of the series. In other words, it is the output list of the last neuron in the series, which is also the last neuron of `NeuronList`. There are also two constraints for this archetype. `AllSingle` expresses that all neurons in the series are single input neurons. The functions `In` and `length` are defined in Coq's list library and define list membership and size of a list, respectively. `SeriesOutput` expresses that the output of the series is equal to the output of the function `SeriesNetworkOutput`. This function takes the input of the series and list of neurons in the series and produces the output of the series. We leave out its definition and just note here that it expresses the details of the input/output connections between the elements of `NeuronList`, and in the degenerate case when `NeuronList` is empty, `NSOutput` is set to the input. (See the definition in our accompanying code.)

The Coq definition of the negative loop is shown below.

```
Record NegativeLoop {Inputs: list nat} :=
MakeNegativeLoop {
  N1: Neuron;
  N2: Neuron;
  NinputN1: (beq_nat (length (Weights N1)) 2%nat)
            = true;
  NinputN2: (beq_nat (length (Weights N2)) 1%nat)
            = true;
  PW1: 0 < (hd 0 (Weights N1));
  PW2: (hd 0 (tl (Weights N1))) < 0;
  PW3: 0 < (hd 0 (Weights N2));
  Connection1: Eq_Neuron2 N1
               (AfterNArch2N1 (ResetNeuron N1)
                              (ResetNeuron N2)
                              Inputs);
  Connection2: Eq_Neuron2 N2
               (AfterNArch2N2 (ResetNeuron N1)
                              (ResetNeuron N2)
                              Inputs)
}.
```

There are only two neurons in this archetype. These neurons are represented by `N1` and `N2` in the definition. The rest of the fields are constraints defining the properties and connections of the archetype. `NinputN1` expresses that `N1` has two inputs. Similarly, `NinputN2` states the number of inputs for `N2`. `N2` is a single input neuron. As mentioned earlier, in Figure 1(d), the solid black circle for an input arrow means that the input has a negative weight and the absence of this circle means that the input has a positive weight. These properties are expressed by constraints `PW1` and `PW2`. In particular, the first input acts as an activator for `N1` and the second one is an inhibitor for `N1`. `PW3` expresses that the only input of `N2` has a positive weight and so is an activator for `N2`. `Connection1` defines the connection of the output of `N2` to the second input of `N1` using the `AfterNArch2N1` function. This function returns a neuron that represents the status of `N1` after applying all inputs in the input list. Again, we leave out the details of the formal definition and refer the reader to the accompanying Coq code. We simply note here that in order to compute this result, three arguments are required—the initial status of `N1`, the initial status of `N2`, and the list of inputs to the archetype. Similarly to the definition of `NeuronSeries`, `Inputs` is an argument of the record `NegativeLoop`. The `Eq_Neuron2` function (whose definition is also omitted, see the code) checks equality of neurons by checking equality of each individual field. `Connection2` defines the connection of the output of `N1` to the only input of `N2` and is defined similarly to `Connection1`. `Connection2` uses `AfterNArch2N2`, which is the same as `AfterNArch2N1` except that it returns `N2`

after applying all inputs in the input list.

10 Properties of Archetypes and their Proofs in Coq

Some of the properties presented in Section 6 concerning neuron and archetype behaviors have already been fully verified in Coq. The most important ones are: the delayer and filter effects for a single neuron (Property 1), the n-delayer effect for a simple series (Property 2), and fixed-point inhibition for an inhibition archetype (Property 6). For illustration, we report on the Coq proofs of the first two, plus provide one supplementary property. We give their complete proofs to show the structure of the main inductions as well as to show other high-level mathematical proof strategies used in the proofs. Many such strategies can be mapped to Coq proof commands called *tactics*; some examples that can be seen in the code include, `induction`, `inversion`, and `destruct`.

In the statement of the properties we present, we omit the assumption that the input sequence of the neuron is a binary list containing only 0s and 1s. It is, of course, taken into account in the Coq code. We adopt many other conventions to enhance readability when stating properties and presenting the corresponding proofs. We state the properties using pretty-printed Coq syntax, with some abbreviations for our own definitions. For instance, we use mathematical fonts and conventions for Coq text, e.g., `(Output N)` is written $Output(N)$, `(Tau N)` is written $\tau(N)$, `(Weights N)` is written $w(N)$, `(Leak_Factor N)` is written $r(N)$, and `(Current N)` is written $p(N)$. In addition, if $w(N)$ is a list of the form $[w_1; \ldots; w_n]$ for some $n \geq 0$, for $i = 1, \ldots, n$, we often write $w_i(N)$ to denote w_i. Furthermore, we use notation and operators from the Coq standard library for lists. For example, *length* and + are list operators; the latter is the notation used here for list concatenation. In addition, although for a neuron N, the list $Output(N)$ is encoded in reverse order in our Coq model, we use forward order when presenting properties and proofs here.

10.13 The Delayer Effect for a Single-Input Neuron

Recall that the delayer effect of Property 1 of Subsection 1 concerns a single neuron having only one input. Since a neuron is in an inactive state initially, its output at time 0 is 0. When a neuron has only one input, and the corresponding weight is greater than or equal to the neuron activation threshold, then the neuron transfers the input sequence to the output without any modification (except for a "delay" of length 1). For example, if a single input neuron receives 0100110101 as its input sequence, it will produce 00100110101 as output. Neurons with this property are mainly just transferring signals. Humans have some of this type of neuron in their auditory system, associated to a chemical synapse. This property is formalized as Property 11.

Property 11 ***[Delayer effect for a single-input neuron]***

$$\forall (N : neuron)(input : list\ nat),$$
$$length(w(N)) = 1 \wedge w_1(N) \geq \tau(N) \rightarrow$$
$$Output(N') = [0] + input$$

In the above statement, N' denotes the neuron obtained by initializing N and then processing the input (using $ResetNeuron$ and repeated applications of $NextNeuron$). Observe that in Definition 1, p is a function of time. Time in our Coq model corresponds to the position in the output list. If $Output(N)$ has length t, then $p(N)$ stores $p(t-1)$ from Definition 1. By applying $NextNeuron$ to N and to the next input obtaining N', we obtain that $Output(N')$ has length $t+1$ and $p(N')$ stores the value $p(t)$ from Definition 1.

In order to prove Property 11, we require the following lemma, which states that when a neuron has one input and the input weight is greater than or equal to the threshold, then the potential value of that neuron is always non-negative.

Lemma 1

$$\forall (N : neuron)(input : list\ nat),$$
$$length(w(N)) = 1 \wedge w_1(N) \geq \tau(N) \rightarrow p(N') \geq 0$$

As previously explained, $p(N')$ is the most recent value of the potential function of neuron N, i.e., the one obtained after processing all the input values. The proof of this lemma is in the accompanying Coq code. We use it here to prove Property 11.

Proof 1 (of Property 11) *The proof is by induction on the length of the input sequence as follows.*

Base case: input = [] (the empty list). If there is no input in the input sequence, the neuron will keep its initial status, i.e., $N = N'$. So, $Output(N') = [0]$. Therefore, $Output(N') = [0] = [0] + [] = [0] + input$.

Induction case: We assume that the property is true for input and we must show that it holds for some $input'$ of the form $(input + [h])$ for some additional input value h. Let N' be the neuron resulting from processing input, and let N'' be the neuron after processing $input'$. By the induction hypothesis, we know $Output(N') = [0] + input$ and we must prove that $Output(N'') = [0] + input'$.

Note that $\tau(N) = \tau(N') = \tau(N'')$ and similar equalities hold for r and w_1, so we use them interchangeably. Because input is a binary list, we know that $h = 0$ or $h = 1$. We break the proof into two different cases, depending on the value of h.

First, we assume that $input' = input + [0]$ and we prove that $Output(N'') = Output(N') + [0]$. In this case,

the most recent input to the neuron is 0. Again, to relate this to Definition 1, let t be the time at which we process the last input. We calculate $p(N'')$, which corresponds to $p(t)$, i.e., the potential value of the neuron at time t; also the value $p(N')$ represents $p(t-1)$ in this definition. Using the first and second clauses of the definition of $p(t)$, respectively, the value is one of:
$p(N'') = w_1(N') \cdot 0 = 0$ *or*
$p(N'') = w_1(N') \cdot 0 + r(N') \cdot p(N')$.

In the first case, $p(N'') = 0$ and we know $0 < \tau(N)$, because $\tau(N)$ is always positive. So, by the second clause of the definition of $y(t)$, the next output of the neuron will be 0. The other case, which comes from the second clause of the definition of $p(t)$, has the same result. In this case, the condition on this clause says that $p(N') < \tau(N')$ and we must show that $p(N'') = r(N') \cdot p(N') < \tau(N)$. Recall that $r(N')$, the leak factor of the neuron, is between 0 and 1. So, multiplying any number, that is less than a positive number, by a value between 0 and 1, gives a value that is smaller than or equal to the original number. Therefore, by the definition of $y(t)$, the next output of the neuron will be 0 again. We can conclude now that by adding 0 to the input sequence, a 0 will be produced in the output. Thus, $Output(N'') = Output(N') + [0]$. Using our induction hypothesis, we have: $Output(N'') = Output(N') + [0] = [0] + input + [0] = 0 + input'$.

Second, we assume that $input' = input + [1]$ and we will prove that $Output(N'') = Output(N') + [1]$. In this case, the most recent input of the neuron is 1. Again, we calculate the potential value of N'' the definition of $p(t)$:
$p(N'') = w_1(N') \cdot 1 = w_1(N')$ *or*
$p(N'') = w_1(N') \cdot 1 + r(N') \cdot p(N') = w_1(N') + r(N') \cdot p(N')$.

In the first case, when $p(N'') = w_1(N')$, we know that $w_1(N) \geq \tau(N)$ by assumption in the statement of the property; we know that $w_1(N) = w_1(N')$ as discussed, and thus $p(N'') \geq \tau(N)$. So by the definition of $y(t)$, the next output of the neuron will be 1. In the second case, $p(N') \geq 0$ according to Lemma 1, and it is always the case that $r(N') \geq 0$, so we can conclude that $r(N') \cdot p(N') \geq 0$. Because $w_1(N) \geq \tau(N)$ and adding a non-negative value to the greater side of an inequality keeps it that way, we can conclude that $p(N'') = w_1(N') + r(N') \cdot p(N') \geq \tau(N)$. Therefore, again by the definition of $y(t)$, the next output of the neuron will be 1 again. Thus, we can conclude in both cases that by adding 1 to the input sequence, a 1 will be produced in the output. Thus, $Output(N'') = Output(N') + [1]$. Using our induction hypothesis, we have: $Output(N'') = Output(N') + [1] = [0] + input + [1] = 0 + input'$.

This completes the proof.

10.14 The Delayer Effect for a Simple Series

Here we revisit the delayer effect for the simple series in Figure 1(a) (Property 2 of Subsection 2). If we have a series of n single input neurons and all of them have the delayer effect, then the output of the whole structure is the input plus n leading zeros. In other words, this structure transfers the input sequence exactly with a delay marked by the n leading zeros, denoted as $zeros(n)$ in the statement of the property below. The `NeuronSeries` record defined in the previous section to represent a simple series is denoted *NeuronSeries* here. This property is expressed as follows.

Property 12 ***[Delayer effect for a simple series]***

$$\forall (input : list\ nat)(Series : (NeuronSeries\ input))(n : nat),$$
$$length(NeuronList) = n \wedge$$
$$\forall\ i = 1, \ldots n,\ length(w(NeuronList[i])) = 1 \wedge$$
$$\forall\ i = 1, \ldots n,\ w_1(NeuronList[i]) > \tau(NeuronList[i]) \rightarrow$$
$$Output = zeros(n) + input$$

In this statement, *Series* is a variable of record type $(NeuronSeries\ input)$, where *input* is the argument to the *NeuronSeries* function, which builds a record. For readability, we abbreviate $(NeuronList\ Series)$, which represents the first field of *Series*, as *NeuronList*. Also, $NeuronList[i]$ denotes the i^{th} neuron in *NeuronList* and *Output* abbreviates $(NSOutput\ Series)$, which is equal to $(Output\ (NeuronList[n]))$

Proof 2 (of Property 12) *The proof this time is by induction on the length of NeurlonList.*

Base case: $N = 0$. This is the degenerate case where there are no neurons in the series and the output of the series is set to the input. (In particular, recall that the `SeriesOutput` *constraint of the* `NeuronSeries` *record is defined using the* `SeriesNetworkOutput` *function, which is defined so that it returns the input list.) Thus in this case, $Output = input = zeros(0) + input$.*

Base case: $N = 1$. In this case, there is only one neuron in the series and we know that this neuron has the delayer effect. According to Property 11 proved earlier, we can conclude that $Output = [0] + input = zeros(1) + input$.

Induction case: We assume that the property holds for NeuronList of length k. Let $Output'$ be the output of this series of length k. Thus by the induction hypothesis, $Output' = zeros(k) + input$. We must show that the property holds for a $NeuronList + [M]$, where M is a neuron such that $length(w(M)) = 1$ and $w_1(M) > \tau(M)$. Let $Output''$ be the output of this series of length $k + 1$. The input sequence for M is the final output of NeuronList, which is $zeros(k) + input$. By the assumptions of this property, all neurons in $NeuronList + [M]$ satisfy Property 11, i.e., have the delayer effect, including the last one M, which means that its output is equal to its input plus a leading 0. In other words, $Output'' = [0] + zeros(k) + input$. Therefore, we can conclude that $Output'' = [0] + zeros(k) + input = zeros(k+1) + input$.

This completes the proof.

10.15 The Spike Decreasing Property for a Single-Input Neuron

The *spike decreasing* property states that a single input neuron cannot produce in its output sequence more 1s than the number of 1s in its input sequence. For example, if the input sequence is 11100110101, which contains seven 1s, the output of the neuron will have less than or equal to seven number of 1s. This property is a consequence of the fact that a single input neuron has either the delayer effect or the filter effect, and both of them do not increase the number of 1s. It is an important property of LI&F neurons and is expressed as property 13.

Property 13 ***[Spike decreasing property for a single-input neuron]***

$$\forall(N : neuron)(input : list\ nat),$$
$$length(w(N)) = 1 \rightarrow$$
$$count(input, 1) \geq count(Output(N'), 1)$$

In this property, *count* is a function that calculates the number of occurrences of the number given as the second argument in the list given as the first argument. So, $count(input, 1)$ returns the number of 1s in the input list and $count(Output(N'), 1)$ computes the number of 1s in the output list of the neuron N'. Recall that here N' is the neuron after initializing N and then applying all the inputs in the input list.

A lemma is needed to prove this property. The following lemma expresses that when a single input neuron receives a 0 as input at any point in time, it cannot produce 1 in the output as follows.

Lemma 2

$$\forall(N : neuron)(input : list\ nat),$$
$$length(w(N)) = 1 \wedge input = [0] \rightarrow last(Output(N')) = 0$$

In the statement of this lemma, *last* represents the last element of a list and N' represents the neuron obtained from N by processing a single 0 as the next input. (In other words, N' is obtained from N by a single application of $NextNeuron$ without first applying $ResetNeuron$.) Thus, $p(N')$ represents the value used to calculate $last(Output(N'))$, which is the last output value of the neuron. The proof of this lemma is a straightforward consequence of Definition 1.

Proof 3 (of Lemma 2) *Since the input is a single* 0*, we know from Definition 1 that* $p(N') = w_1(N) \cdot 0$ *or* $p(N') = w_1(N) \cdot 0 + r(N) \cdot p(N)$*. Simplifying the multipication by* 0*, we have these two cases:* $p(N') = w_1(N) \cdot 0 = 0$ *or* $p(N') = w_1(N) \cdot 0 + r(N) \cdot p(N) = r(N) \cdot p(N)$*. We know that* $\tau(N') = \tau(N) > 0$*. Thus, in the first case,* $p(N') = 0 < \tau(N')$*. According to Definition 1, the next output of the neuron will be 0 because its potential value* $p(N')$ *is less than its activation threshold* $\tau(N')$*.*

In the second case, when $p(N') = r(N) \cdot p(N)$*, we know that* $p(N) < \tau(N) = \tau(N')$*. Also, for every neuron, the leak factor is between 0 and 1. In other words,* $0 \leq r(N) = r(N') \leq 1$*. If* $p(N) < 0$*, multiplying it by a positive number will produce a negative value which is less than* $\tau(N)$*. So,* $p(N') = r(N) \cdot p(N) < 0 < \tau(N) = \tau(N')$*. On the other hand, if* $p(N) \geq 0$*, multiplying it by a number between 0 and 1, will produce a smaller number. So,* $p(N') = r(N) \cdot p(N) < p(N) < \tau(N) = \tau(N')$*. As can be seen,* $p(N') < \tau(N')$ *in this case also and thus the next output of the neuron will be 0. This completes the proof.*

We can now use this lemma to prove property 13.

Proof 4 (of Property 13) *The proof is by induction on the length of the input sequence as follows.*

Base case: input = [] *(the empty list). If there is no input in the input sequence, the neuron will keep its initial status, i.e.,* $N = N'$*. So,* $Output(N') = [0]$*. Therefore,* $count(input, 1) = count([], 1) = 0 \geq count(Output(N'), 1) = count([0], 1) = 0$*.*

Induction case: We assume that the property is true for input and we must show that it holds for some $input'$ *of the form* $(input + [h])$ *for some additional input value* h*. Let* N' *be the neuron resulting from processing input, and let* N'' *be the neuron after processing* $input'$*. By the induction hypothesis, we know* $count(input, 1) \geq count(Output(N'), 1)$ *and we must prove that* $count(input', 1) \geq count(Output(N''), 1)$*.*

Because input is a binary list, we know that $h = 0$ *or* $h = 1$*. We break this into two different cases, depending on the value of* h*.*

First, we assume that $input' = input + [0]$*. Because 0 is added to the input list, we have* $count(input', 1) = count(input, 1)$*. According to Lemma 2, that we just proved, having 0 as the last input in* $input'$ *cannot produce 1 as the last output in the output list. So, we can conclude that* $Output(N'') = Output(N') + [0]$*. Similarly, because 0 is added to the output,* $count(Output(N'), 1) = count(Output(N''), 1)$*. Therefore,* $count(input', 1) = count(input, 1) \geq count(Output(N'), 1) = count(Output(N''), 1)$*.*

Second, we assume that $input' = input + [1]$*. In this case, 1 is added to the input list. Because the number of 1s is increased, it is not important that this new input produce 0 or 1 in the output list. The number of 1s in the output list will remain unchanged or increase by 1. So,* $count(Output(N''), 1)$ *is at most* $count(Output(N'), 1) + 1$*. Also, we know that* $count(input', 1) = count(input + [1], 1) = count(input, 1) + 1$*. By the induction hypothesis we know that,* $count(input, 1) \geq count(Output(N'), 1)$*. By adding 1 to both sides of the inequality, we will have* $count(input, 1) + 1 \geq count(Output(N'), 1) + 1$*. Therefore,* $count(input, 1) + 1 = count(input', 1) \geq count(Output(N''), 1)$*.*

This completes the proof.

11 Comparison of the Proposed Approaches and Future Work

In this paper we proposed two formal approaches to study some key properties concerning the dynamic evolution of neurons, some basic archetypes, and some archetype couplings. These formal approaches are original and complementary with respect to the main international projects aiming at understanding the human brain, such as the Human Brain Project [56], which is mainly based on large systems of differential equations.

We provide a full implementation for both approaches. As far as the model checking approach is concerned, the Lustre code for all the archetypes and properties can be found at

`https://redmine.i3s.unice.fr/projects/archetypes/repository`.

As far as the theorem proving approach is concerned, as mentioned, the Coq code for the archetype implementation, properties, and proofs is given at

`http://www.site.uottawa.ca/~afelty/coq-archetypes/`.

First of all, we should underline that both approaches are likely to be improved. Concerning the model checking technique, before checking the validity of a property, we actually fix an interval of values for each parameter and we ask to the model checker whether the property holds for the chosen parameter set. We are aware of a research group working on the integration of Linear Decision Diagrams (LDD) within the model checker. This would allow to automatically infer parameter sets for which properties are verified. Concerning the theorem proving technique, although the proofs we have completed require some sophisticated reasoning, there is still a significant amount that is common between them. As we continue, we expect to encounter more complex inductions as we consider more complex properties. Thus, it will become important to automate as much of the proofs as possible, most likely by writing *tactics* tailored to the kind of induction, case analysis, and mathematical reasoning that is needed here. Furthermore, defining general relations that can be specialized to specific patterns will likely also be very useful for the kinds of properties that are important for more complex networks.

In absolute terms, we could not say one of the two approaches presented here is strongly preferable with respect to the other one for the formal study of the dynamic properties of neuron archetypes. The main advantage of the model checking methodology is that it is completely automatic: once a given property has been correctly encoded, the user just needs to press a button to know whether the property is verified or not. So the presence of an expert is not needed to obtain a proof. Another strength of model checking is that, in case a property does not hold in a given model, a counter-example is automatically provided. Such an execution trace can give hints in understanding what should be modified in the system so that the property is satisfied. As another advantage, we should underline that for our application domain model checking is not time consuming. In fact, even if the transition system corresponding to each model is exponential with respect to the number of variables, we could get an immediate answer for all our properties because the chosen model checker, kind2 [24], exploits some advanced features to improve scalability, such as *modular reasoning* (each node can be assigned its own properties and verified individually. The results of the verification process can be reused in the analysis of other components calling that node).

On the other hand, model checkers often cannot prove properties at the desired level of generality. As a matter of fact, the use of a proof assistant guarantees that the properties we prove are true in the *general case*, such as true for any input values, any length of input, and any amount of time. As an example, let us consider the simple series. With Lustre, we were able to write a node which encodes the expected behavior of the circuit. Then, we could call the kind2 model checker to test whether the property at issue is valid for some input series with a fixed length. With Coq we can prove that the desired behavior is true whatever the length and the parameters of the series are. Another advantage of the theorem proving approach is that, since we have access to proofs, a successful proof of a given property can be exploited to prove similar properties. The drawbacks are that proofs can be long and an expert is needed to perform them.

Coming back to the biological issue that motivated our study, our results until now show that archetype coupling can either *modulate* the behaviors displayed by the single archetypes (e.g. extend an oscillation period) or clearly give rise to *new behaviors*. Furthermore, several different couplings turn out to display the same behavior (whatever their input sequences are). As a next step, we intend to make a systematic study of all the possible archetypes and couplings, even including several archetypes, and classify their respective properties. The number of constrained graphs of a few elements and, concomitantly, the number of elementary behaviors is reduced, both from a logical and biological standpoint. To use again the analogy of words, few syllables allow to build thousands of currently used words and a quasi-infinity of sentences. Moreover, beyond a certain number of neurons and above all a certain level of connectivity, other functional processes different from archetypes are supposed to emerge, like cell assemblies [57, 58]. So the composition process should stop after a finite and limited (compared to the

size of the brain) number of iterations. We should thus rapidly fall back on already studied behaviors, but expressed through various instances of neuronal networks, reducible to few neuronal archetypes. This means that, at a certain point, this approach should allow to express any relevant neuronal network as an archetype (a basic one or the result of a concatenation procedure).

Our feeling is that, to perform a formal advanced analysis of archetypes and their composition, the model checking and theorem proving approaches should be used together in a pragmatic way. When trying to prove a given property, the idea is to first test its validity for some crucial given parameter intervals using model checking, and eventually refine the model thanks to the provided counter-examples so that the property holds in the defined context. Once some key tests have passed, the theorem proving technique can be exploited to prove the property in a more general context. The complementarity of these two powerful techniques could make us advance rapidly in the study of the fundamental structural and functional properties of the elementary bricks of brain and cognition.

Acknowledgements We thank Alexandre Muzy for fruitful discussions on neuron modeling.

References

1. Olaf Sporns. The human connectome: Origins and challenges. *NeuroImage*, 80:53 – 61, 2013. Mapping the Connectome.
2. Olaf Sporns. Structure and function of complex brain networks. *Dialogues in clinical neuroscience*, 15:247–262, 2013.
3. Olaf Sporns. Graph theory methods: applications in brain networks. *Dialogues in clinical neuroscience*, 20:111–121, 2018.
4. Alex Fornito, Andrew Zalesky, and Michael Breakspear. Graph analysis of the human connectome: Promise, progress, and pitfalls. *NeuroImage*, 80:426 – 444, 2013. Mapping the Connectome.
5. Kiyotoshi Matsuoka. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological cybernetics*, 56(5-6):345–353, 1987.
6. E. De Maria, A. Muzy, D. Gaffé, A. Ressouche, and F. Grammont. Verification of temporal properties of neuronal archetypes modeled as synchronous reactive systems. In Eugenio Cinquemani and Alexandre Donzé, editors, *Hybrid Systems Biology - 5th International Workshop, HSB 2016, Grenoble, France, October 20-21, 2016, Proceedings*, pages 97–112, 2016.
7. Elisabetta De Maria, Thibaud L'Yvonnet, Daniel Gaffé, Annie Ressouche, and Franck Grammont. Modelling and formal verification of neuronal archetypes coupling. In *Proceedings of the 8th International Conference on Computational Systems-Biology and Bioinformatics, Nha Trang City, Viet Nam, December 7-8, 2017*, pages 3–10, 2017.
8. Abdorrahim Bahrami, Elisabetta De Maria, and Amy Felty. Modelling and verifying dynamic properties of biological neural networks in coq. In *Proceedings of the 9th International Conference on Computational Systems-Biology and Bioinformatics*, CSBio 2018, pages 12:1–12:11, New York, NY, USA, 2018. ACM.
9. Wulfram Gerstner and Werner Kistler. *Spiking Neuron Models: An Introduction*. Cambridge University Press, New York, NY, USA, 2002.
10. W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
11. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.
12. David R. Gilbert and Monika Heiner. Advances in computational methods in systems biology. *Theor. Comput. Sci.*, 599:2–3, 2015.
13. François Fages, Sylvain Soliman, and Nathalie Chabrier-rivier. Modelling and querying interaction networks in the biochemical abstract machine biocham. *Journal of Biological Physics and Chemistry*, 4:64–73, 2004.
14. Adrien Richard, Jean-Paul Comet, and Gilles Bernot. Graph-based modeling of biological regulatory networks: Introduction of singular states. In *Computational Methods in Systems Biology, International Conference, CMSB 2004, Paris, France, May 26-28, 2004, Revised Selected Papers*, pages 58–72, 2004.
15. Elisabetta De Maria, François Fages, Aurélien Rizk, and Sylvain Soliman. Design, optimization and predictions of a coupled model of the cell cycle, circadian clock, DNA repair system, irinotecan metabolism and exposure control under temporal logic constraints. *Theor. Comput. Sci.*, 412(21):2108–2127, 2011.
16. Carolyn L. Talcott and Merrill Knapp. Explaining response to drugs using pathway logic. In *Computational Methods in Systems Biology - 15th International Conference, CMSB 2017, Darmstadt, Germany, September 27-29, 2017, Proceedings*, pages 249–264, 2017.
17. N. Halbwachs. Synchronous programming of reactive systems. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1998.
18. N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*, Twente, June 1993. Workshops in Computing, Springer Verlag.
19. N. Halbwachs and P. Raymond. Validation of synchronous reactive systems: from formal verification to automatic testing. In *ASIAN'99, Asian Computing Science Conference*, Phuket (Thailand), December 1999. LNCS 1742, Springer Verlag.
20. B. Jeannet. Dynamic partitioning in linear relation analysis. application to the verification of reactive systems. *Formal Methods in System Design*, 23(1):5–37, 2003.
21. Luke webpage. `http://www.it.uu.se/edu/course/homepage/pins/vt11/lustre`.
22. A. Franzén. Using satisfiability modulo theories for in-

ductive verification of lustre programs. *Electr. Notes Theor. Comput. Sci.*, 144(1):19–33, 2006.
23. G. Hagen and C. Tinelli. Scaling up the formal verification of lustre programs with smt-based techniques. In *2008 Formal Methods in Computer-Aided Design*, pages 1–9, Nov 2008.
24. Adrien Champion, Alain Mebsout, Christoph Sticksel, and Cesare Tinelli. The kind 2 model checker. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 510–517, 2016.
25. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.
26. S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.
27. Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.
28. René Thomas, Denis Thieffry, and Marcelle Kaufman. Dynamical behaviour of biological regulatory networks—i. biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bulletin of Mathematical Biology*, 57(2):247–276, Mar 1995.
29. Venkatramana N. Reddy, Michael L. Mavrovouniotis, and Michael N. Liebman. Petri net representations in metabolic pathways. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology, Bethesda, MD, USA, July 1993*, pages 328–336, 1993.
30. Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Proceedings of the 6th Pacific Symposium on Biocomputing, PSB 2001, Hawaii, USA, January 3-7, 2001*, pages 459–470, 2001.
31. Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.
32. Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, François Fages, and Vincent Schächter. Modeling and querying biomolecular interaction networks. *Theor. Comput. Sci.*, 325(1):25–44, 2004.
33. R. Hofestädt and S. Thelen. Quantitative modeling of biochemical networks. *In Silico Biology*, 1:39–53, 1998.
34. Rajeev Alur, Calin Belta, and Franjo Ivancic. Hybrid modeling and simulation of biomolecular networks. In *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, pages 19–32, 2001.
35. Andrew Phillips and Luca Cardelli. Efficient, correct simulation of biological processes in the stochastic pi-calculus. In *Computational Methods in Systems Biology, International Conference, CMSB 2007, Edinburgh, Scotland, September 20-21, 2007, Proceedings*, pages 184–199, 2007.
36. Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.
37. Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model verifier. In *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, pages 495–499, 1999.
38. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 585–591, 2011.
39. Elisabetta De Maria, Joëlle Despeyroux, and Amy P. Felty. A logical framework for systems biology. In *Formal Methods in Macro-Biology - First International Conference, FMMB 2014, Nouméa, New Caledonia, September 22-24, 2014. Proceedings*, pages 136–155, 2014.
40. M. Dénès, B. Lesage, Y. Bertot, and A. Richard. Formal proof of theorems on genetic regulatory networks. In *11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 69–76, Sep. 2009.
41. Adnan Rashid, Osman Hasan, Umair Siddique, and Sofiane Tahar. Formal reasoning about systems biology using theorem proving. *PLOS ONE*, 12(7):1–27, 07 2017.
42. Bruno Pinaud, Oana Andrei, Maribel Fernández, Hélène Kirchner, Guy Melançon, and Jason Vallet. PORGY : a visual analytics platform for system modelling and analysis based on graph rewriting. In *17ème Journées Francophones Extraction et Gestion des Connaissances, EGC 2017, 24-27 Janvier 2017, Grenoble, France*, pages 473–476, 2017.
43. W. Maas. Networks of spiking neurons: The third generation of neural network models. *Trans. Soc. Comput. Simul. Int.*, 14(4):1659–1671, December 1997.
44. H. Paugam-Moisy and S. M. Bohte. Computing with spiking neuron networks. In *Handbook of Natural Computing*, pages 335–376. 2012.
45. G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.
46. E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, Sept 2004.
47. A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.
48. L. Lapicque. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J Physiol Pathol Gen*, 9:620–635, 1907.
49. Bogdan Aman and Gabriel Ciobanu. Modelling and verification of weighted spiking neural systems. *Theoretical Computer Science*, 623:92 – 102, 2016.
50. Elisabetta De Maria and Cinzia Di Giusto. Parameter learning for spiking neural networks modelled as timed automata. In *Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2018) - Volume 3: BIOINFORMATICS, Funchal, Madeira, Portugal, Jan-*

uary 19-21, 2018., pages 17–28, 2018.
51. D. Purves, G. J. Augustine, D. Fitzpatrick, W. C. Hall, A.S. LaMantia, J. O. McNamara, and S. M. Williams, editors. *Neuroscience*. Sinauer Associates, Inc., 3rd edition, 2006.
52. E. De Maria, A. Muzy, D. Gaffé, A. Ressouche, and F. Grammont. Verification of Temporal Properties of Neuronal Archetypes Using Synchronous Models. Research Report 8937, UCA, Inria ; UCA, I3S ; UCA, LEAT ; UCA, LJAD, July 2016.
53. *Coq reference manual. Retrieved from https://coq.inria.fr/distrib/current/-refman/index.html*.
54. Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Inf. Comput.*, 76(2/3):95–120, 1988.
55. Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, 2012.
56. Egidio D'Angelo, Giovanni Danese, Giordana Florimbi, Francesco Leporati, Alessandra Majani, Stefano Masoli, Sergio Solinas, and Emanuele Torti. The human brain project: High performance computing for brain cells hw/sw simulation and understanding. In *2015 Euromicro Conference on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28, 2015*, pages 740–747, 2015.
57. D. O. Hebb. Organization of behavior. new york: Wiley. *The Journal of Physiology*, 6(307):335, 1949.
58. F. Grammont and A. Riehle. Spike synchronization and firing rate in a population of motor cortical neurons in relation to movement direction and reaction time. *Biological Cybernetics*, 88(5):360–373, May 2003.

Chapter 7

A Model-checking Approach to Reduce Spiking Neural Networks

Elisabetta De Maria, Daniel Gaffé, Cédric Girard Riboulleau, and Annie Ressouche. *9th International Conference on Bioinformatics Models, Methods and Algorithms, 2018*

A Model-checking Approach to Reduce Spiking Neural Networks

Elisabetta De Maria[1], Daniel Gaffé[2], Cédric Girard Riboulleau[3] and Annie Ressouche[3]
[1]*Univ. Côte d'Azur, CNRS, I3S, UMR 7271,06900 Sophia Antipolis, France*
[2]*Univ. Côte d'Azur, CNRS, LEAT, UMR 7248, 06900 Sophia Antipolis, France*
[3]*Univ. Côte d'Azur, INRIA SAM,06902 Sophia-Antipolis, France*
edemaria@i3s.unice.fr, Daniel.GAFFE@unice.fr, cedric.girard-riboulleau@inria.fr, annie.ressouche@inria.fr

Keywords: Neural Spiking Networks, Probabilistic Models, Temporal Logic, Model Checking, Network Reduction.

Abstract: In this paper we formalize Boolean Probabilistic Leaky Integrate and Fire Neural Networks as Discrete-Time Markov Chains using the language PRISM. In our models, the probability for neurons to emit spikes is driven by the difference between their membrane potential and their firing threshold. The potential value of each neuron is computed taking into account both the current input signals and the past potential value. Taking advantage of this modeling, we propose a novel algorithm which aims at reducing the number of neurons and synaptical connections of a given network. The reduction preserves the desired dynamical behavior of the network, which is formalized by means of temporal logic formulas and verified thanks to the PRISM model checker.

1 INTRODUCTION

Since a few decades, neurobiologists and bioinformatics researchers work in concert to model neural networks, aiming at understanding the interactions among neurons, and the way they participate in the different vital functions of human beings. Models become bigger and bigger, and the necessity of reducing them while preserving their expected dynamics emerges, especially in the scope of the obtention of models which are suitable for formal verification. In this paper we tackle the issue of neural network reduction.

In the literature neural network modeling is often classified into three generations [Maass, 1997, Paugam-Moisy and Bohte, 2012]. First generation models, represented by McCulloch-Pitts one [McCulloch and Pitts, 1943], deal with discrete inputs and outputs and their computational units are a set of logic gates with a threshold activation function. Second generation models, whose most known one is the multi-layer perceptron [Cybenko, 1989], exploit real valued activation functions. These networks, whose real-valued outputs represent neuron firing rates, are extensively used in the domain of artificial intelligence and are also known as artificial neural networks. Third generation networks, also called *Spiking Neural Networks* [Paugam-Moisy and Bohte, 2012], stand out for the relevance of time aspects. Precise spike firing times are taken into account. Furthermore, they consider not only current input spikes but also past ones. In [Izhikevich, 2004], Spiking Neural Networks are classified with respect to their biophysical plausibility, that is, to the number of behaviors (i.e., typical responses to an input pattern) they can display. Among these models, the Hodgkin-Huxley model [Hodgkin and Huxley, 1952] is the one able to reproduce most behaviors. However, its simulation process is really expensive even for a few neurons.

In this work we choose to rely on the Leaky Integrate and Fire (LI&F) model [Lapicque, 1907], a computationally efficient approximation of a single-compartment model. Our LI&F model is augmented with probabilities. More precisely, the probability for neurons to emit spikes is driven by the difference between their membrane potential and their firing threshold. Probabilistic neurons are encoded as Discrete-Time Markov Chains thanks to the modeling language at PRISM user's disposition. PRISM [Kwiatkowska et al., 2011] is a tool that allows not only to model different probabilistic systems (with discrete or continuous time, with or without nondeterminism), but also to specify their expected behavior thanks to the use of temporal logics, a formalism for describing the dynamical evolution of systems. Furthermore, PRISM provides a model checker [Clarke et al., 1999], which is a tool for automatically verifying whether a given system satisfies or not a property

expressed in temporal logic. In case the property does not hold in the system, the user can have access to a counter-example, that is, an execution trace falsifying the property at issue, which often helps in finding modifications in the model for the property to be satisfied. In order to apply model-checking techniques efficiently, the model to handle should be as small as possible.

Taking advantage of this modeling and verification framework, we introduce a novel algorithm which aims at reducing the number of neurons and synaptical connections of a given neural network. The proposed reduction preserves the desired dynamical behavior of the network, which is formalized by means of temporal logic formulas and verified thanks to the PRISM model checker. More precisely, a neuron is removed if its suppression has a low impact on the probability for a given temporal logic formula to hold. Observe that, other than their utility in lightening models, algorithms for neural network reduction have a forthright application in the medical domain. In fact, they can help in detecting weakly active (or inactive) zones of the human brain.

The issue of reducing biological networks is not new in systems biology. Emblematic examples can be found in [Naldi et al., 2011], where the authors propose a methodology to reduce regulatory networks preserving some dynamical properties of the original models, such as stable states, in [Gay et al., 2010], where the authors study model reductions as graph matching problems, or in [Paulevé, 2016], whose author considers finite-state machines and proposes a technique to remove some transitions while preserving all the (minimal) traces satisfying a given reachability property. As far as neural networks are concerned, to the best of our knowledge the core of the existing reduction approaches only deals with second generation networks. Several methods to train a network that is larger than necessary and then remove the superfluous parts, known as pruning techniques, are explained in [Reed, 1993] . Finally, in [Menke and Martinez, 2009] the authors introduce an oracle learning methodology, which consists in using a larger model as an oracle to train a smaller model in order to obtain a smaller acceptable model. With oracle learning, the smaller model is created initially and trained using the larger model, whereas with pruning, connections are removed from the larger model until the desired size is reached.

The paper is organized as follows. In Section 2 we introduce a probabilistic version of the Leaky Integrate and Fire Model. Section 3 is devoted to the PRISM modeling language and the temporal logic PCTL (Probabilistic Computation Tree Logic). In Section 4 we describe our modeling of neural networks as Discrete-Time Markov Chains in PRISM. Finally, in Section 5 we introduce the novel algorithm for the reduction of neuronal networks, and in Section 6 we discuss some future research directions.

2 PROBABILISTIC LEAKY INTEGRATE AND FIRE MODEL

We model neuron networks as Boolean Spiking Networks, where the electrical properties of neurons are represented through the Leaky Integrate and Fire (LI&F) model. In this modeling framework, neural networks are seen as directed graphs whose nodes stand for neurons and whose edges stand for synaptical connections. Edges are decorated with weights: positive (resp. negative) weights represent activations (resp. inhibitions). The dynamics of each neuron is characterized through its *(membrane) potential value*, which represents the difference of electrical potential across the cell membrane. At each time unit, the potential value is computed taking into account present input spikes and the previous decayed potential value. In order to weaken the past potential value, it is multiplied by a *leak factor*. In our probabilistic LI&F model, the probability for each neuron to emit an action potential, or *spike*, is governed by the difference between the potential value and a given *firing threshold*. For positive (resp. negative) values of this difference, the more its absolute value is big, the more (resp. the less) is the probability to emit a spike. After each spike emission, the neuron potential is reset to zero. In the literature, other ways exist to incorporate probabilities in LI&F models, such as the Noisy Integrate and Fire models [Di Maio et al., 2004, Fourcaud and Brunel, 2002], where a noise is added to the computation of the potential value.

More formally, we give the following definitions for probabilistic LI&F networks.

Definition 1 (Boolean Probabilistic Spiking Integrate and Fire Neural Network)**.** A Boolean Probabilistic Integrate and Fire Neural Network *is a tuple* (V, E, w)*, where:*

- V *are Boolean probabilistic spiking integrate and fire neurons,*
- $E \subseteq V \times V$ *are synapses,*
- $w : E \to \mathbb{Q} \cap [-1, 1]$ *is the synapse weight function associating to each synapse* (u, v) *a weight* w_{uv}*.*

We distinguish three disjoint sets of neurons: V_i *(input neurons),* V_{int} *(intermediary neurons), and* V_o *(output neurons), with* $V = V_i \cup V_{int} \cup V_o$*.*

Definition 2 (Boolean Probabilistic Spiking Integrate and Fire Neuron)**.** A Boolean Probabilistic Spiking Integrate and Fire Neuron v *is a tuple* (τ, r, p, y)*, where:*

- $\tau \in \mathbb{N}$ *is the* firing threshold*,*
- $r \in \mathbb{Q} \cap [0,1]$ *is the* leak factor*,*
- $p : \mathbb{N} \to \mathbb{Q}_0^+$ *is the [membrane]* potential *function defined as*

$$p(t) = \begin{cases} \sum_{i=1}^{m} w_i \cdot x_i(t), & \text{if } p(t-1) \geqslant \tau \\ \sum_{i=1}^{m} w_i \cdot x_i(t) + r \cdot p(t-1), & \text{otherwise} \end{cases}$$

where $p(0) = 0$*, m is the number of inputs of the neuron v, w_i is the weight of the synapse connecting the i^{th} input neuron of v to the neuron v, and $x_i(t) \in \{0,1\}$ is the signal received at the time t by the neuron v through its i^{th} input synapse (observe that, after the potential overcomes its threshold, it is reset to 0),*

- $y : \mathbb{N} \to \{0,1\}$ *is the output function of the neuron.*

Supposing to discretize $p(t) - \tau$ in $k+1$ positive intervals and $k+1$ negative intervals, the probability for the neuron v to emit a spike can be described as follows:

$$P(y(t) = 1) = \begin{cases} 1 & \text{if } p(t) - \tau \geq l_k \\ p_{2k} & \text{if } l_{k-1} \leq p(t) - \tau < l_k \\ \vdots & \\ p_{k+1} & \text{if } 0 \leq p(t) - \tau < l_1 \\ p_k & \text{if } -l_1 \leq p(t) - \tau < 0 \\ \vdots & \\ p_1 & \text{if } -l_k \leq p(t) - \tau < -l_{k-1} \\ 0 & \text{if } p(t) - \tau < -l_k \end{cases} \tag{1}$$

with $\{l_1, \ldots, l_k\} \subseteq \mathbb{N}^+$ *such that* $l_i < l_{i+1}$ $\forall i \in \{1, \ldots, k-1\}$ *and* $\{p_1, \ldots, p_{2k}\} \subseteq]0,1] \cap \mathbb{Q}$ *such that* $p_i < p_{i+1}$ $\forall i \in \{1, \ldots, 2k-1\}$.

In our implementation, the probability values are chosen in order to conform to a sigmoidal function.

3 THE PROBABILISTIC MODEL CHECKER PRISM

The probabilistic model checker PRISM [Kwiatkowska et al., 2011] is a tool for formal modeling and analysis of systems with a random or probabilistic behavior. It supports several types of probabilistic models: discrete ones, namely discrete-time Markov chains, Markov decision processes, and probabilistic automata, and continuous ones, namely continuous-time Markov chains, probabilistic timed automata, and priced probabilistic timed automata. In this work we rely on discrete-time Markov chains, which are transition systems augmented with probabilities. Their set of states represent the possible configurations of the system being modeled, and transitions between states model the evolution of the system, which occurs in discrete-time steps. Probabilities of making transitions between states are given by discrete probability distributions. Markov chains are memoryless, that is, their current state contains all the information needed to compute future states (*Markov property*). More precisely, the following definition can be given:

Definition 3 (Discrete-Time Markov Chain)**.** *A Discrete-Time Markov Chain (DTMC) over a set of atomic propositions AP is a tuple (S, S_{init}, P, L) where:*

- *S is a set of states (state space)*
- *$S_{init} \subseteq S$ is the set of initial states*
- *$P : S \times S \to [0,1]$ is the transition probability matrix, where $\sum_{s' \in S} P(s,s') = 1$ for all $s \in S$*
- *$L : S \to 2^{AP}$ is a function labeling states with atomic propositions over AP.*

An example of DTMC representing a simplified neuron is graphically depicted in Figure 1, where the neuron can be either active or inactive.

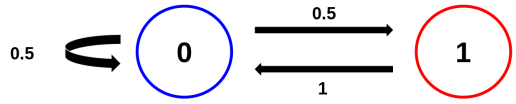


Figure 1: **Example of a two-state DTMC representing a simplified neuron.** When the neuron is inactive (state 0), it remains inactive with a probability of 0.5, and it becomes active, and thus emits a spike (state 1), with a probability of 0.5. When it is active, is becomes inactive with a probability of 1.

3.1 The PRISM Modeling Language

PRISM provides a state-based modeling language inspired from the reactive modules formalism of [Alur and Henzinger, 1999]. A model is composed by a set of *modules* which can interact with each other. At each moment, the state of each module is given by the values of its *local variables*, and the global state of the whole model is determined by the local state of all its modules. The dynamics of each module is described by a set of commands of the form:

$$[]guard \to prob_1 : update_1 + \ldots + prob_n : update_n;$$

where *guard* is a predicate over all the variables of the model, indicating the condition to be verified in order

to execute the command, and each *update* indicates a possible transitions of the model, to be achieved by giving new values to the variables of the module. Each update is assigned to a probability and, for each command, the sum of probabilities must be 1. The PRISM code for the DTMC of Figure 1 is given in Figure 2. In such a simple module, the square brackets at the beginning of each command are empty but it is possible to add labels representing *actions*. These actions can be used to force two or more modules to make transitions simultaneously. In this work, we take advantage of this feature to synchronize neurons in networks. Finally, PRISM models can be extended

```
dtmc // Discrete-Time Markov Chain

module simplified_neuron
// Declaration of the integer variable y
y: [0..1] init 0;
// Commands
[] y=0 -> 0.5:(y'=0) + 0.5:(y'=1);
[] y=1 -> 1:(y'=0);
endmodule
```

Figure 2: **PRISM code for the DTMC of Figure 1.** The only variable y, representing the state of the neuron, ranges over $[0..1]$. Its initial value is 0. When the guard is $y = 0$, the updates $(y' = 0)$ and $(y' = 1)$ and their associated probabilities state that the value of y remains at 0 with probability 0.5 and passes to 1 with probability 0.5. When $y = 1$, the variable changes its value to 0 with a probability of 1.

with *rewards* [Kwiatkowska et al., 2007], which allow to associate real values to states or transitions of models. As an example, in Figure 3 we show how to augment the simplified neuron code of Figure 2 in order to add a reward each time the neuron is active, and thus to count the number of spike emissions.

```
rewards "y"
y=1: 1;
endrewards
```

Figure 3: **Addition of a reward to the PRISM code for a simplified neuron.** Each time $y = 1$ (spike emission), the reward increases of one time unit.

3.2 Probabilistic Temporal Logic

PRISM allows to specify the dynamics of DTMCs thanks to the temporal logic PCTL (Probabilistic Computation Tree Logic) introduced in [Hansson and Jonsson, 1994], which extends the logic CTL (Computation Tree Logic) [Clarke et al., 1986] with time and probabilities. The following state quantifiers are available in PCTL: **X** (next time), which specifies that a property holds at the next state of a given path, **F** (sometimes in the future), which requires a property to hold at some state on the path, **G** (always in the future), which imposes that a property is true at every state on the path, and **U** (until), which holds if there is a state on the path where the second of its argument properties holds and, at every preceding state on the path, the first of its two argument properties holds. Note that the classical path quantifiers **A** (forall) and **E** (exist) of CTL are replaced by probabilities. *Thus, instead of saying that some property holds for all paths or for some paths, we can express that a property holds for a certain fraction of the paths* [Hansson and Jonsson, 1994]. The most important operator in PCTL is **P**, which allows to reason about the probability of event occurrences. The property `P bound [prop]` is true in a state s of a model if the probability that the property *prop* is satisfied by the paths from state s satisfies the bound *bound*. As an example, the PCTL property `P= 0.5 [X (y = 1)]` holds in a state if the probability that $y = 1$ is true in the next state equals 0.5. All the state quantifiers given above, with the exception of X, have bounded variants, where a time bound is imposed on the property. For example, the property `P> 0.9 [F<=10 y=1]` is true in a state if the probability of y being equal to 1 within 10 time units is greater than 0.9. Furthermore, in order to compute the actual probability that some behavior of a model is displayed, the P operator can take the form `P=? [prop]`, which evaluates to a numerical rather than to a Boolean value. As an example, the property `P =? [G (y = 0)]` expresses the probability that y is always equal to 0.

PRISM also allows to formalize properties which relate to the expected values of rewards. This is possible thanks to the **R** operator, which can be used in the following two forms: `R bound [rewardprop]`, which is true in a state of a model if the expected reward associated with *rewardprop* when starting from that state meets the bound, and `R=? [rewardprop]`, which returns the actual expected reward value. Some specific operators are introduced in PRISM in order to deal with rewards. In the rest of the paper, we mainly exploit **C** (cumulative-reward). The property `C<=t` corresponds to the reward cumulated along a path until t time units have elapsed. As an example, consider the reward y of Figure 3. The property `R{"y"} =? [C <= 100]` returns the expected value of the reward y within 100 time units. PRISM provides model-checking algorithms [Clarke et al., 1999] to automatically validate DTMCs over PCTL properties and reward-based ones. The available algorithms are able to compute the actual probability that some behavior of a model is displayed, when required.

4 NEURONAL NETWORKS IN PRISM

This section is devoted to the modeling of Boolean Probabilistic LI&F neural networks as DTMCs using the PRISM language. As a first step, we introduce an *input generator* module in order to generate input sequences on the alphabet $\{0,1\}$ for input neurons. Such a module deals with one only Boolean variable whose value is 1 (resp. 0) in case of spike (resp. no spike) emission. We provide several instantiations of such a module to be able to generate different kinds of input sequences: persistent ones (containing only 1), oscillatory ones, and random ones.

We then define a *neuron* module to encode LI&F neurons. The state of each neuron is characterized by two variables: a first Boolean variable denoting the spike emission, and a second integer[1] variable representing the potential value, computed as a function of the current inputs and the previous potential value, as shown in Definition 2. The difference between the potential value and the firing threshold is discretized into $k+1$ positive intervals and $k+1$ negative intervals and a PRISM command is associated to each one of this intervals: if the neuron is inactive and the difference between the potential and the threshold meets the guard, the neuron is activated with a certain probability p (a bigger difference gives a higher probability). The neuron remains inactive with a probability equal to $1-p$. The aforecited commands share the same label (`to`). To complete the modeling of a single neuron, we add a command acting when the neuron is active: with a probability of 1 it resets the potential value and makes the neuron inactive. Such a command is connoted by a new different label (`reset`), that turns out to be useful to synchronize the output of neurons with the input of the following ones. Thanks to a module renaming feature at PRISM user's disposition, neurons with different parameters can be easily obtained starting from the standard neuron module.

We then model the synaptical connections between the neurons of the network. In order to avoid a neuron to be reset before its successor takes its activation into account, we introduce a *transfer* module consisting of one only variable ranging over $\{0,1\}$ and being initialized at 0. Thanks to synchronization labels, at each `reset` of the first neuron this variable passes to 1 first, and then goes back to 0, synchronizing with the second neuron (label `to`), which takes the received signal into account to compute its potential value.

[1] We exploit integer rather than rational numbers in order to have efficient model-checking performances

The PRISM code for the neuron and the transfer modules can be found in [Girard Riboulleau, 2017], where the PRISM model has been validated against several PCTL properties.

5 MODEL-CHECKING BASED REDUCTION ALGORITHM

In this section we introduce a novel algorithm for the reduction of Boolean Probabilistic LI&F networks. The algorithm supposes the network to be implemented as a DTMC in PRISM [Girard Riboulleau, 2017] and makes several calls to the PRISM model checker, in order to retrieve the probability relative to the satisfaction of some PCTL formulas and the value of some rewards.

As a first step, the algorithm aims at identifying, and thus removing, the *wall* neurons, that is, the neurons that are not able emit, even if they receive a persistent sequence of spikes as input. More formally, a neuron can be characterized as a *wall* one if its probability to be always quiescent (inactive) is 1: `P=1 [G (y=0)]`. When the algorithm detects a *wall* neuron, it removes not only the neuron but also its descendants whose only incoming synaptical connection comes, directly or indirectly [2], from this neuron, and its ancestors whose only outgoing edge enters, directly or indirectly, the neuron.

As a second step, the algorithm aims at testing whether the suppression of the remaining neurons preserves or not the dynamics of the network. The removal of a neuron (and its associated ancestors and descendants) is authorized if the following two quantities are kept (modulo a certain error):

Quantitative criterion. The reward computing the number of emitted spikes (within 100 time units) of each output neuron.

Qualitative criterion. The probability for a given PCTL property (concerning the output neurons) to hold.

The number of spikes emitted by a neuron within 100 time units can be computed thanks to the following reward-based PRISM property: `R{"y"} =? [C <= 100]`. An example of key property concerning the qualitative behavior of a neuron is the following one, expressing an oscillating trend: `P=? [G((y=1 => (y=1 U y=0))& (y=0 => (y=0 U y=1)))]`. Such a formula requires every spike emission to be followed by a quiescent state (not necessarily immediately) and viceversa.

[2] By one only edge or a path

Formulas comparing the behaviors of several neurons can be written as well. Observe that the respect of both quantitative and qualitative criteria is needed for a neuron removal. In fact, the output neurons of two different networks could exhibit the same spike rate but display a completely different behavior. On the other hand, their could exhibit the same qualitative behavior (e.g., an oscillatory trend), but have quite different spike rates.

The pseudo-code for the proposed reduction algorithm is given in Algorithm 1. It takes as input a Boolean Probabilistic LI&F network $G = (V, E, w)$ as given in Definition 1, a PCTL property *Prop* concerning the dynamical behavior of the output neurons of the network, and an allowed error value ε. Only intermediary neurons are affected by the reduction process, that is, input and output neurons cannot be removed. Intermediary neurons are first visited following a depth first search (DFS) in order to remove *wall* neurones (and their associated ancestors and descendants). We opt for a depth first visit instead of a breath first visit to avoid expensive backtracking. Furthermore, DFS is better suited to our approach because it allows to quickly take into account all the descendants of a node and cut them if necessary.

The procedure in charge to remove a neuron (and its associated ancestors and descendants) is *REMOVAL* (see Algorithm 2). Another depth first traversal of the (remaining) intermediary neurons is then performed to identify (and thus remove thanks to the *REMOVAL* procedure) neurons whose removal has a low influence (according to ε) on the probability for *Prop* to be satisfied and on the rate spike. It is possible to see that, for each traversal, each edge is visited at most twice, once forward and once backward.

An example of application of the algorithm to a neural network composed of eight neurons and nine edges is graphically depicted in Figure 4. The reduction process leads to a reduced network consisting of four neurons and three edges.

In Table 1 we consider several neural networks composed of four neurons (with only one input and output neuron) and their corresponding reduced network, consisting of three neurons. For each network, we give the spike rate of the output neuron, and the number of states and transitions of the corresponding PRISM transition system. For an average error lower than 0.65 in the spike rate, we have an average reduction of the state number of a factor 19.6 and an average reduction of the transition number of a factor 19.64 when passing from the complete to the reduced network.

Algorithm 1 LI&F REDUCTION (G, *Prop*, ε)

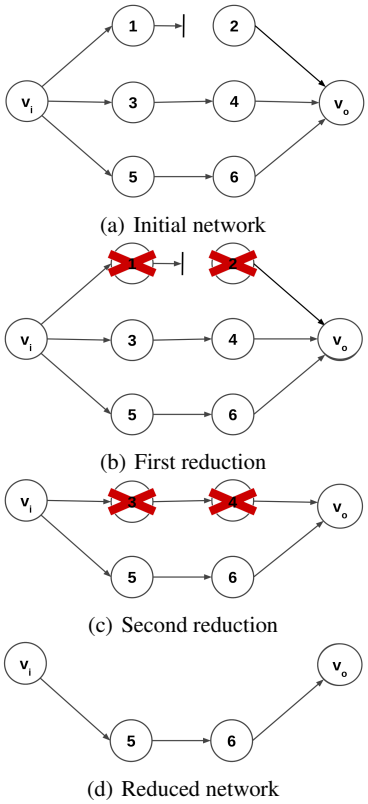
$$We distinguish input, intermediary, and output neurons
Let $V = V_i \bigcup V_{int} \bigcup V_o$
for all $v_i \in V_{int}$ in a DFS visit **do**
    Set to 1 all the input signals of v_i
    $$ Call to the PRISM model checker
    if $P = 1[G(y_i = 0)]$ $$ y_i is the output of v_i **then**
        REMOVAL(v_i) $$ V_{int} and E are modified by *REMOVAL*
Set to 1 all the input signals of the neurons of V_i
for all $v_o \in V_o$ **do**
    Compute $r_o = spike\ rate\ of\ v_o$ thanks to PRISM rewards
Compute $p = Prob(Prop\ is\ TRUE)$ thanks to PRISM
for all $v_i \in V_{int}$ **do**
    Let $V' = V \setminus \{v_i\}$, $E' = E \setminus \{(v_i, v_j) \cup (v_k, v_i)\}$ s.t. $v_j, v_k \in V$
    Let $G' = (V', E', w)$
    for all $v_o \in V_o$ **do**
        Compute $r'_o = spike\ rate\ of\ v_o$ in G' thanks to PRISM rewards
    Compute $p' = Prob(Prop\ is\ TRUE)$ in G' thanks to PRISM
    if $|r'_o - r_o| \leq \varepsilon$ for all $v_o \in V_o$ **and** $|p' - p| \leq \varepsilon$ **then**
        REMOVAL(v_i)

Complete network (4 neurons)			**Reduced network (3 neurons)**		
spikes	states	transitions	spikes	states	transitions
6.75	252 820	565 068	6.72	16 816	37 510
7.93	148 480	319 549	7.92	10 255	22 029
6.11	291 002	653 121	6.07	19 903	44 559
9.55	280 719	608 641	9.54	12 062	26 109
6.75	225 169	500 245	6.72	15 005	33 254
5.86	265 683	591 546	5.48	19 571	43 542
8.75	149 641	325 890	8.65	9 418	20 466
7.39	193 897	425 447	7.37	12 951	28 341
8.16	961 701	2 142 739	8.16	21 829	48 547
5.84	60 422	129 583	5.84	7 239	15 516
9.50	196 433	427 391	9.50	12 768	27 762
10.26	333 952	715 179	10.26	10 841	23 101
8.13	192 456	424 269	8.13	12 634	27 809
7.77	**273 260**	**602 205**	**7.72**	**13 946**	**30 657**

Table 1: **Comparative table of some complete and reduced networks.** We consider 13 different (with different parameters) networks composed of four neurons and the corresponding networks obtained thanks to the reduction algorithm. All the reduced networks consist of three neurons. We give spike rates of output neurons, and the number of states and transitions of the corresponding PRISM transition systems. The last row refers to averages.

Algorithm 2 REMOVAL(v_i)

1: \$\$ It suppresses a neuron v_i and all the neurons only having v_i as ancestor or descendant
2: $V_{int} = V_{int} \setminus \{v_i\}$
3: $E = E \setminus \{(v_i, v_j) \cup (v_k, v_i)\}$ s.t. $v_j, v_k \in V$
4: $G_{work} = (V_{int}, E)$
5: **for all** v_j descendant of v_i in a DFS visit of G **do**
6: \$\$ If v_j has no incoming edges in G_{work}, we remove v_j and its exiting edges
7: **if** $\{(v_h, v_j)\} = \varnothing$ **then**
8: $V_{int} = V_{int} \setminus \{v_j\}$
9: $E = E \setminus \{(v_j, v_h)\}$ s.t. $v_h \in V$
10: **for all** v_k ancestors of v_i in a DFS visit of G **do**
11: \$\$ If v_k has no outgoing edges in G_{work}, we remove v_k and its incoming edges
12: **if** $\{(v_k, v_l)\} = \varnothing$ **then**
13: $V_{int} = V_{int} \setminus \{v_k\}$
14: $E = E \setminus \{(v_l, v_k)\}$ s.t. $v_l \in V$
15: $G = G_{work}$



(a) Initial network

(b) First reduction

(c) Second reduction

(d) Reduced network

Figure 4: **Application of the reduction algorithm on a neuronal network of eight neurons.** The only input neuron is v_i and the only output neuron is v_o. The other neurons are numbered according to a DFS order. The neuron 1 is identified as a *wall* one. The first reduction step (4(b)) consists in removing the neuron 1 and, consequently, the neuron 2, because its only ingoing edge comes from the neuron 1. No other *wall* neuron is detected. The second reduction (4(c)) is due to the fact that the removal of neuron 4 influences neither the satisfaction of *Prop* nor the spike emission rate of v_o. The neuron 3 is also removed because its only output edge enters the neuron 4. The final reduced network is given in 4(d).

6 CONCLUSIONS

In this paper we have formalized Boolean Probabilistic Leaky Integrate and Fire Neural Networks as Discrete-Time Markov Chains using the language PRISM. Taking advantage of this modeling, we have proposed a novel algorithm which aims at reducing the number of neurons and synaptical connections of a given network. The reduction preserves the desired dynamical behavior of the output neurons of the network, which is formalized by means of temporal logic formulas and verified thanks to the PRISM model checker.

This work is the starting point for several future research directions. From a modeling point of view, the use of labels entails a break time during which the different modules communicate but all the other functions are stopped. Namely, in our model the `reset` label causes a break time after each spike emission. A big number of spikes leads thus to a big number of break times and we intend to minimize these times. We also plan to model the refractory time of neurons, a lapse of time following the spike emission during which the neuron is not able to emit (even if it continues to receive signals). At this aim, we may need to take advantage of Probabilistic Timed Automata, which are at PRISM user's disposition.

Concerning the reduction algorithm, for the moment we show our approach to be efficient for small networks, i.e., the removal of only one neuron drastically reduces the size of the transition system. As for future work, we intend to scale our methodology.

The actual version of the reduction algorithm finds a reduction which conforms to the expected behavior of the network. We find one solution among several possible ones, but this solution is not necessary the optimal one, that is, it does not necessarily minimize the difference of behavior between the complete and the reduced network. In order to help the research of optimal solutions, we intend to perform a sensitivity analysis of our networks, aiming at identifying the parameters playing a most important role in the verification of some given temporal properties.

ACKNOWLEDGEMENTS

We thank Alexandre Muzy for fruitful discussions on neuron behaviors.

REFERENCES

Alur, R. and Henzinger, T. (1999). Reactive modules. *Formal Methods in System Design*, 15(1):7–48.

Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263.

Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model checking*. MIT press.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Di Maio, V., Lansky, P., and Rodriguez, R. (2004). Different types of noise in leaky integrate-and-fire model of neuronal dynamics with discrete periodical input. *General physiology and biophysics*, 23:21–38.

Fourcaud, N. and Brunel, N. (2002). Dynamics of the firing probability of noisy integrate-and-fire neurons. *Neural computation*, 14(9):2057–2110.

Gay, S., Soliman, S., and Fages, F. (2010). A graphical method for reducing and relating models in systems biology. *Bioinformatics*, 26:i575–i581.

Girard Riboulleau, C. (2017). Modèles probabilistes et vérification de réseaux de neurones. Master's thesis, Université Nice-Sophia-Antipolis.

Hansson, H. and Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544.

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Transaction On Neural Networks*, 15(5):1063–1070.

Kwiatkowska, M., Norman, G., and Parker, D. (2007). Stochastic model checking. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 220–270. Springer.

Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G. and Qadeer, S., editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer.

Lapicque, L. (1907). Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J Physiol Pathol Gen*, 9:620–635.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Menke, J. E. and Martinez, T. R. (2009). Artificial neural network reduction through oracle learning. *Intelligent Data Analysis*, 13(1):135–149.

Naldi, A., Remy, E., Thieffry, D., and Chaouiya, C. (2011). Dynamically consistent reduction of logical regulatory graphs. *Theorical Computer Science*, 412:2207–2218.

Paugam-Moisy, H. and Bohte, S. M. (2012). Computing with spiking neuron networks. In *Handbook of Natural Computing*, pages 335–376.

Paulevé, L. (2016). Goal-oriented reduction of automata networks. In *Computational Methods in Systems Biology - 14th International Conference, CMSB 2016, Cambridge, UK, September 21-23, 2016, Proceedings*, pages 252–272.

Reed, R. (1993). Pruning algorithms-a survey. *Trans. Neur. Netw.*, 4(5):740–747.

Bibliography

[1] Emna Ben Abdallah, Tony Ribeiro, Morgan Magnin, Olivier Roux, and Katsumi Inoue. Inference of Delayed Biological Regulatory Networks from Time Series Data. In Ezio Bartocci, Pietro Lio, and Nicola Paoletti, editors, *14th International Conference on Computational Methods for Systems Biology (CMSB 2016)*, volume 9859 of *Computational Methods in Systems Biology*, Cambridge, United Kingdom, September 2016. Springer.

[2] Annabelle Ballesta, Sandrine Dulong, Chadi Abbara, Boris Cohen, Alper Okyar, Jean Clairambault, and Francis Levi. A combined experimental and mathematical approach for molecular-based optimization of irinotecan circadian delivery. *PLOS Computational Biology*, 7:1–12, 09 2011.

[3] Annabelle Ballesta, Pasquale F. Innominato, Robert Dallmann, David A. Rand, and Francis A. Lévi. Systems chronotherapeutics. *Pharmacological Reviews*, 69(2):161–199, 2017.

[4] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995.

[5] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[6] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. Bionetgen: software for rule-based modeling of signal trans-

duction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004.

[7] Ralf Blossey, Luca Cardelli, and Andrew Phillips. A compositional approach to the stochastic dynamics of gene networks. *Transactions in Computationial Systems Biology*, 3939:99–122, 2006.

[8] Luca Bortolussi and Alberto Policriti. Hybrid systems and biology: Continuous and discrete modeling for systems biology. In *Proceedings of the Formal Methods for the Design of Computer, Communication, and Software Systems 8th International Conference on Formal Methods for Computational Systems Biology*, SFM'08, pages 424–448, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Dario Campagna and Carla Piazza. Hybrid automata, reachability, and systems biology. *Theor. Comput. Sci.*, 411(20):2037–2051, 2010.

[10] Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, François Fages, and Vincent Schächter. Modeling and querying biomolecular interaction networks. *Theor. Comput. Sci.*, 325(1):25–44, 2004.

[11] Claudine Chaouiya. Petri net modelling of biological networks. *Briefings in Bioinformatics*, 8(4):210–219, 07 2007.

[12] A. Ciliberto, B. Novak, and J. J. Tyson. Steady states and oscillations in the p53/mdm2 network. *Cell Cycle*, 4(3):488–93, 2005.

[13] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model verifier. In *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, pages 495–499, 1999.

[14] Federica Ciocchetta and Jane Hillston. Bio-pepa: A framework for the modelling and analysis of biological systems. *Theor. Comput. Sci.*, 410(33-34):3065–3084, August 2009.

[15] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[16] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[17] Sarp Coskun, A. Ercument Cicek, Nicola Lai, Ranjan Dash, Zehra Ozsoyoglu, and Gultekin Ozsoyoglu. An online model composition tool for system biology models. *BMC systems biology*, 7:88, 09 2013.

[18] Egidio D'Angelo, Giovanni Danese, Giordana Florimbi, Francesco Leporati, Alessandra Majani, Stefano Masoli, Sergio Solinas, and Emanuele Torti. The human brain project: High performance computing for brain cells hw/sw simulation and understanding. In *2015 Euromicro Conference on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28, 2015*, pages 740–747, 2015.

[19] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.

[20] Joëlle Despeyroux and Kaustuv Chaudhuri. A hybrid linear logic for constrained transition systems. In *19th International Conference on Types for Proofs and Programs, TYPES 2013, April 22-26, 2013, Toulouse, France*, pages 150–168, 2013.

[21] E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier, 1995.

[22] François Fages and Aurélien Rizk. On temporal logic constraint solving for analyzing numerical data time series. *Theor. Comput. Sci.*, 408(1):55–65, 2008.

[23] François Fages and Sylvain Soliman. Formal cell biology in biocham. In *Formal Methods for Computational Systems Biology, 8th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2008, Bertinoro, Italy, June 2-7, 2008, Advanced Lectures*, pages 54–80, 2008.

[24] François Fages, Sylvain Soliman, and Nathalie Chabrier-Rivier. Modelling and querying interaction networks in the biochemical abstract machine biocham. *Journal of Biological Physics and Chemistry*, 4:64–73, 2004.

[25] Timur Fayruzov, Jeroen Janssen, Dirk Vermeir, Chris Cornelis, and Martine De Cock. Modelling gene and protein regulatory networks with answer set programming. *IJDMB*, 5(2):209–229, 2011.

[26] Laurent Fiack, Benoit Miramond, and Laurent Rodriguez. A neural processing unit for self-organizing maps. In *Workshop on "Neuromorphic and Brain-Based Computing Systems"*, Grenoble, France, Mar 2015.

[27] Maxime Folschette, Loïc Paulevé, Katsumi Inoue, Morgan Magnin, and Olivier F. Roux. Identification of biological regulatory networks from process hitting models. *Theor. Comput. Sci.*, 568:49–71, 2015.

[28] Steven Gay, Sylvain Soliman, and FranÃğois Fages. A graphical method for reducing and relating models in systems biology. *Bioinformatics*, 26:i575–i581, 2010.

[29] M. Gebser, A. Konig, T. Schaub, S. Thiele, and P. Veber. The bioasp library: Asp solutions for systems biology. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 383–389, Oct 2010.

[30] Wulfram Gerstner and Werner Kistler. *Spiking Neuron Models: An Introduction*. Cambridge University Press, New York, NY, USA, 2002.

[31] David R. Gilbert and Monika Heiner. Advances in computational methods in systems biology. *Theor. Comput. Sci.*, 599:2–3, 2015.

[32] M. Gill, S. McKeever, and D. Gavaghan. Model composition for biological mathematical systems. In *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 217–224, 2014.

[33] Judah Goldfeder and Hillel Kugler. Temporal logic based synthesis of experimentally constrained interaction networks. In *Molecular Logic and Computational Synthetic Biology - First International Symposium, MLCSB 2018, Santiago, Chile, December 17-18, 2018, Revised Selected Papers*, pages 89–104, 2018.

[34] N. Halbwachs. Synchronous programming of reactive systems. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1998.

[35] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994.

[36] Monika Heiner, R Donaldson, and David Gilbert. *Petri Nets for Systems Biology, in Iyengar, M.S. (ed.), Symbolic Systems Biology: Theory and Methods; Chapter 21.* 01 2010.

[37] R. Hofestädt and S. Thelen. Quantitative modeling of biochemical networks. *In Silico Biology*, 1:39–53, 1998.

[38] Gerard J. Holzmann. *The SPIN Model Checker - primer and reference manual*. Addison-Wesley, 2004.

[39] Michael Hucka. Systems biology markup language (SBML). In *Encyclopedia of Computational Neuroscience*. 2014.

[40] Trey Ideker, Timothy Galitski, and Leroy Hood. A new approach to decoding life: Systems biology. *Annual Review of Genomics and Human Genetics*, 2(1):343–372, 2001. PMID: 11701654.

[41] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, Sept 2004.

[42] C.I. Hong J. Zámborszky and A.C. Nag. Computational analysis of mammalian cell division gated by a circadian clock: Quantized cell cycles and cell size. *Journal of Biological Rhythms*, 22(6):542âĂŞ–553, 2007.

[43] Sumit Jha and Christopher Langmead. Exploring behaviors of stochastic differential equation models of biological systems using change of measures. *BMC bioinformatics*, 13 Suppl 5:S8, 04 2012.

[44] Stuart Kauffman. Homeostasis and differentiation in random genetic control networks. *Nature*, 224:177–178, 1969.

[45] Haseong Kim and Erol Gelenbe. Stochastic gene expression modeling with hill function for switch-like gene responses. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 9(4):973–979, 2012.

[46] Hiroaki Kitano. Systems biology: A brief overview. *Science*, 295(5560):1662–1664, 2002.

[47] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 585–591, 2011.

[48] L. Lapicque. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J Physiol Pathol Gen*, 9:620–635, 1907.

[49] Jean-Christophe Leloup and Albert Goldbeter. Toward a detailed computational model for the mammalian circadian clock. *Proceedings of the National Academy of Sciences*, 100(12):7051–7056, 2003.

[50] Oded Maler and Grégory Batt. Approximating continuous systems by timed automata. In *Formal Methods in Systems Biology, First International Workshop, FMSB 2008, Cambridge, UK, June 4-5, 2008. Proceedings*, pages 77–89, 2008.

[51] Elisabetta De Maria, François Fages, Aurélien Rizk, and Sylvain Soliman. Design, optimization and predictions of a coupled model of the cell cycle, circadian clock, DNA repair system, irinotecan metabolism and exposure control under temporal logic constraints. *Theor. Comput. Sci.*, 412(21):2108–2127, 2011.

[52] Elisabetta De Maria, Daniel Gaffé, Cédric Girard Riboulleau, and Annie Ressouche. A model-checking approach to reduce spiking neural networks. In *Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2018) - Volume 3: BIOINFORMATICS, Funchal, Madeira, Portugal, January 19-21, 2018.*, pages 89–96, 2018.

[53] Elisabetta De Maria, Cinzia Di Giusto, and Laetitia Laversa. Spiking neural networks modelled as timed automata: with parameter learning. *Journal of Natural Computing*, 2019.

[54] Elisabetta De Maria, Thibaud L'Yvonnet, Daniel Gaffé, Annie Ressouche, and Franck Grammont. Modelling and formal verification of neuronal archetypes coupling. In *Proceedings of the 8th International Conference on Computational Systems-Biology and Bioinformatics, Nha Trang City, Viet Nam, December 7-8, 2017*, pages 3–10, 2017.

[55] Elisabetta De Maria, Thibaud L'Yvonnet, Sabine Moisan, and Jean-Paul Rigault. Probabilistic activity recognition for serious games with applications in medicine. In *Proceedings of the Seventh International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'19), Shenzhen, China, November 9, 2019.* Springer.

[56] Julien Martinelli, Jeremy Grignard, Sylvain Soliman, and François Fages. A Statistical Unsupervised Learning Algorithm for Inferring Reaction Networks from Time Series Data. In *ICML 2019 Workshop on Computational Biology*, Long Beach, CA, United States, June 2019.

[57] Hiroshi Matsuno, Masao Nagasaki, and Satoru Miyano. Hybrid petri net based modeling for biological pathway simulation. *Natural Computing*, 10(3):1099–1120, 2011.

[58] Kiyotoshi Matsuoka. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological cybernetics*, 56(5-6):345–353, 1987.

[59] Auréliens Naldi, Elisabeth Remy, Denis Thieffry, and Claudine Chaouiya. Dynamically consistent reduction of logical regulatory graphs. *Theorical Computer Science*, 412:2207–2218, 2011.

[60] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

[61] Carlos Olarte, Elaine Pimentel, and Vivek Nigam. Subexponential concurrent constraint programming. *Theor. Comput. Sci.*, 606:98–120, 2015.

[62] Loïc Paulevé. Goal-oriented reduction of automata networks. In *Computational Methods in Systems Biology - 14th International Conference, CMSB 2016, Cambridge, UK, September 21-23, 2016, Proceedings*, pages 252–272, 2016.

[63] Andrew Phillips and Luca Cardelli. Efficient, correct simulation of biological processes in the stochastic pi-calculus. In *Computational Methods in Systems Biology, International Conference, CMSB 2007, Edinburgh, Scotland, September 20-21, 2007, Proceedings*, pages 184–199, 2007.

[64] Adnan Rashid, Osman Hasan, Umair Siddique, and Sofiane Tahar. Formal reasoning about systems biology using theorem proving. *PLOS ONE*, 12(7):1–27, 07 2017.

[65] Venkatramana N. Reddy, Michael L. Mavrovouniotis, and Michael N. Liebman. Petri net representations in metabolic pathways. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology, Bethesda, MD, USA, July 1993*, pages 328–336, 1993.

[66] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.

[67] Aviv Regev and Ehud Shapiro. *The π-calculus as an Abstraction for Biomolecular Systems*, pages 219–266. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[68] Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Proceedings of the 6th Pacific Symposium on Biocomputing, PSB 2001, Hawaii, USA, January 3-7, 2001*, pages 459–470, 2001.

[69] A. Richard. Negative circuits and sustained oscillations in asynchronous automata networks. *Advances in Applied Mathematics*, 44(4):378–392, 2010.

[70] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. In James A. Anderson and Edward Rosenfeld, editors, *Neurocomputing: Foundations of Research*, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[71] Gonzalo A. Ruz, Ana Zúñiga, and Eric Goles. A boolean network model of bacterial quorum-sensing systems. *IJDMB*, 21(2):123–144, 2018.

[72] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[73] J. Sjöström and W. Gerstner. Spike-timing dependent plasticity. *Scholarpedia*, 5(2), 2010.

[74] Tamàs Székely and Kevin Burrage. Stochastic simulation in systems biology. *Computational and Structural Biotechnology Journal*, 12(20):14 – 25, 2014.

[75] Carolyn L. Talcott. The pathway logic formal modeling system: Diverse views of a formal representation of signal transduction. In *IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2016, Shenzhen, China, December 15-18, 2016*, pages 1468–1476, 2016.

[76] Carolyn L. Talcott and Merrill Knapp. Explaining response to drugs using pathway logic. In *Computational Methods in Systems Biology - 15th International Conference, CMSB 2017, Darmstadt, Germany, September 27-29, 2017, Proceedings*, pages 249–264, 2017.

[77] René Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563–585, December 1973.

[78] René Thomas, Denis Thieffry, and Marcelle Kaufman. Dynamical behaviour of biological regulatory networks—i. biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bulletin of Mathematical Biology*, 57(2):247–276, Mar 1995.

[79] Minh Khue Phan Tran, François Brémond, and Philippe Robert. Assistance for older adults in serious game using an interactive system. In *Games and Learning Alliance - 4th Int. Conf (GALA 2015)*, 2015.

[80] Santiago Videla, Carito Guziolowski, Federica Eduati, Sven Thiele, Martin Gebser, Jacques Nicolas, Julio Saez-Rodriguez, Torsten Schaub, and Anne Siegel. Learning boolean logic models of signaling networks with ASP. *Theor. Comput. Sci.*, 599:79–101, 2015.

[81] Santiago Videla, Julio Saez-Rodriguez, Carito Guziolowski, and Anne Siegel. Caspo: a toolbox for automated reasoning on the response of logical signaling networks families. *Bioinformatics*, 33(6):947–950, 2017.

[82] Rui-Sheng Wang, Assieh Saadatpour, and Réka Albert. Boolean modeling in systems biology: an overview of methodology and applications. *Physical Biology*, 9(5):055001, sep 2012.

[83] John Wilson-Kanamori, Vincent Danos, Ty Thomson, and Ricardo Honorato-Zimmer. Kappa Rule-Based Modelling in Synthetic Biology. In Mario Andrea Marchisio, editor, *Computational Methods in Synthetic Biology*, volume 1244 of *Methods in Molecular Biology*, pages 105–135. Springer, 2015.