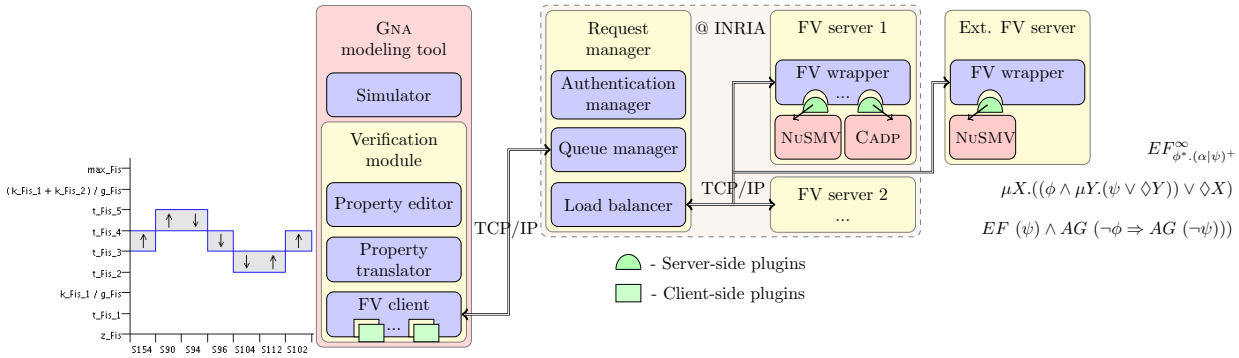
Towards an integrative approach for the modeling and formal verification of biological regulatory networks

Pedro Tiago Gonçalves Monteiro

UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

UNIVERSITÉ CLAUDE BERNARD
LYON 1



Towards an integrative approach for the modeling and formal verification of biological regulatory networks

Pedro Tiago Gonçalves Monteiro
(Mestre)

Dissertação para obtenção do Grau de Doutor em co-tutela
Engenharia Informática e de Computadores

Orientadores:	Doutora	Ana Teresa Correia de Freitas
	Doutor	Hidde de Jong
Co-orientator:	Doutor	Radu Mateescu

Júri

Presidente:	Presidente do Conselho Científico do IST	
Vogais:	Doutor	Jaco van de Pol
	Doutor	Hidde de Jong
	Doutor	Radu Mateescu
	Doutora	Claudine Chaouiya Chantegrel
	Doutora	Ana Teresa Correia de Freitas
	Doutora	Maria Inês Camarate de Campos Lynce de Faria

Maio de 2010

Para a *Sílvia*

Acknowledgments

I wish to thank everyone who made this work possible, but specifically:

- For the opportunity given to me to attend the PhD Program on Computational Biology at Instituto Gulbenkian de Ciência, where I had the chance of receiving courses on the main aspects of computational biology directly by those who where doing the research on each of the subjects lectured.
- To all the co-authors of the articles and posters included in this thesis: Estelle Dumas, Bruno Besson, Hidde de Jong, Radu Mateescu, Miguel Teixeira, Ana Teresa Freitas, Paulo Dias, Delphine Ropers, Arlindo Oliveira, Isabel Sá-Correia and Michel Page.
- To Michel Page, for his brilliant skepticism and pragmatical approach on problems, and for the interesting discussions passed around the white-board.
- To Radu Mateescu, for having accepted to co-supervise this thesis and being my tutor through the world of temporal logics and model checking.
- To Ana Teresa Freitas, for her constant support since the creation of the KDBIO team. For all her patience and help in solving most of the problems that many times overshadowed this thesis, and motivation whenever I was lacking it.
- I am deeply grateful to Hidde de Jong, for trusting me and accepting me in his team for three years. For helping me, carefully proofreading all the manuscripts and always contributing with important suggestions that always increased their quality. Finally, for his inspiration towards research and for pushing me forward whenever I felt most lost.
- To all the current and past members of the IBIS team, Genostar company and other INRIA teams located at Montbonnot for the countless hours of interesting discussions about research or society at large: Valentina Baldazzi, Yves-Pol Danielou, Sara Berthomieux, Sophie Huet, Estelle Dumas, Bruno Besson, Grégory Batt, Alain Viari, Anne Morgat, Eric Coissac, Françoise de Coninck, Hans Geiselmann, François Rechenmann, Pierre-Emmanuel Ciron, Mohammed Youcef, Edouard Blondeau, Adrien Maudet, Evelyne Altariba, Vasil Khalidov, Eugenio Cinquemani, Delphine Ropers, Michel Page and Hidde de Jong.
- And finally, to the all the members of the jury for having accepted the task of judging this thesis, and especially the reporters Claudine Chaouiya and Jaco van de Pol for their valuable comments.

Abstract

The study of biological networks has led to the development of increasingly large and detailed models. While whole-cell models are not on the horizon yet, complex networks underlying specific cellular processes have been modeled in detail. The study of these models by means of analysis and simulation tools leads to large amounts of predictions, typically time-courses of the concentration of several dozens of molecular components in a variety of physiological conditions and genetic backgrounds. This raises the question how to make sense of these simulations, that is, how to obtain an understanding of the way in which particular molecular mechanisms control the cellular process under study, and how to identify interesting predictions of novel phenomena that can be confronted with experimental data.

Formal verification techniques based on model-checking provide a powerful technology to keep up with this increase in scale and complexity. The basic idea underlying model checking is to specify dynamical properties of interest as statements in temporal logic, and to use model-checking algorithms to automatically and efficiently verify whether the properties are satisfied or not by the model. The application of model-checking techniques is hampered, however, by several key issues described in this thesis.

First, the systems biology domain brought to the fore a few properties of the network dynamics that are not easily expressed using classical temporal logics, like Computation Tree Logic (CTL) and Linear Time Logic (LTL). On the one hand, questions about multistability are important in the analysis of biological regulatory networks, but difficult (or impossible) to express in LTL. On the other hand, CTL is capable of dealing with branching time, important for multistability and other properties of non-deterministic models, but it does not do a good job when faced with questions about cycles in a Kripke structure. Second, the problem of posing relevant and interesting questions is critical in modeling in general, but even more so in the context of applying model-checking techniques, due to the fact that it is not easy for non-experts to formulate queries in temporal logic. Finally, most of the existing modeling and simulation tools are not capable of applying model-checking techniques in a transparent way. In particular, they do not hide from the user the technical details of the installation of the model checker, the export in a suitable format of the model and the query, the call of the model checker, and the import of the results produced by the model checker (the true/false verdict and witness/counterexample).

In order to address these issues, in this thesis we propose three different approaches to tackle each of the issues. First, we propose a new temporal logic, called Computation Tree Regular Logic (CTRL), powerful enough to capture the biological properties of multistability (branching-time) and oscillations (linear-time). CTRL extends CTL with regular expressions and fairness operators, achieving a good compromise between expressive power, user-friendliness and complexity of model checking. Second, we propose a new pattern system, based on frequently-asked questions posed by modelers in systems biology. Each pattern is

a high-level query template formulated in structured natural language rather than temporal logic, providing an automatic translation into temporal logic formulas. Third, we propose a service-oriented architecture capable of integrating modeling and simulation tools with current model-checking tools, providing a common environment for the modeling and the verification of biological regulatory networks. This architecture is generic and modular, capable of integrating different model checkers through the use of a plugin system and permitting the connection from different modeling and simulation tools.

The proposed methods were implemented in a new version of the tool Genetic Network Analyzer (GNA 7.0), through the development of a new verification module. This module provides a graphical user interface capable of helping the user with the specification of biological properties, as well as with the visualization and interpretation of verification results. In order to illustrate the use and applicability of the developed methods, GNA 7.0 has been used for the validation of two complex biological models. The first is a model of the network of regulators controlling the carbon starvation response in the bacterium *Escherichia coli*. The second is a model of the activation of the 12-spanner H^+ drug antiporter encoding gene *FLR1* in the presence of the fungicide mancozeb in the eukaryote *Saccharomyces cerevisiae*. Both models were checked for inconsistencies by confronting the model predictions with available experimental data. The identified inconsistencies were subject to further experiments and/or to model revisions.

Formal verification methods are promising tools for upscaling the analysis of biological regulatory networks. The implementation of the proposed approaches into a single integrative environment provide the modeler with a scalable and automated method for analysis of these networks. This thesis helps lower the obstacles to the use of formal verification technology in biology, opening the way for new perspectives like the exploration of symbolic approaches to further scale the size and complexity of the models, and the definition of diagnostics for the semi-automatic revision of the models.

Keywords: systems biology, biological regulatory networks, qualitative simulation, formal verification, temporal logic, model checking.

Resumo

O estudo de redes biológicas tem originado o desenvolvimento de modelos cada vez mais complexos e detalhados. Apesar de ainda não ser possível ter modelos integrais de uma célula, têm sido modeladas em detalhe redes complexas correspondendo a processos celulares específicos. O estudo destes modelos utilizando ferramentas de análise e simulação origina grandes quantidades de previsões, que correspondem normalmente à evolução temporal da concentração de dezenas de componentes celulares numa variedade de condições fisiológicas e genéticas. Isto levanta a questão de como interpretar esta quantidade de simulações, ou seja, como compreender a forma como mecanismos moleculares específicos controlam um dado processo celular, e como identificar previsões interessantes de novos fenómenos que possam ser comparados com dados experimentais.

As técnicas de verificação formal baseadas em *model-checking* são consideradas uma tecnologia poderosa, permitindo acompanhar o aumento de escala e complexidade do problema. A ideia por detrás da técnica de *model-checking* consiste na especificação de propriedades de interesse como formulas em lógica temporal, e na utilização de algoritmos de *model-checking* para verificar de forma automática e eficaz se estas propriedades são ou não satisfeitas pelo modelo. No entanto, a aplicação destas técnicas tem sido dificultada por um conjunto importante de factores, descritos e analisados no âmbito desta tese.

Em primeiro lugar, na área científica da biologia de sistemas têm sido tratadas diversas questões, tais como a identificação e a caracterização de propriedades dinâmicas de uma rede biológica que não são facilmente expressas usando lógicas temporais clássicas, como as lógicas *Computation Tree Logic* (CTL) e *Linear Time Logic* (LTL). Por um lado, questões acerca da multi-estabilidade são importantes na análise de redes biológicas, sendo no entanto difíceis (ou impossíveis) de expressar usando a lógica LTL. Por outro lado, a lógica CTL é suficientemente expressiva para lidar com a ramificação temporal, condição necessária para a multi-estabilidade e outras propriedades de modelos não-determinísticos, mas não é adequada para a expressão de oscilações. Em segundo lugar, o problema de como elaborar perguntas relevantes é essencial no processo de modelação em geral, mas ainda mais relevante no contexto da aplicação de técnicas de *model-checking* à validação de modelos biológicos, devido ao facto de não ser fácil para o utilizador comum formular questões em lógica temporal. Em terceiro lugar, a maioria das ferramentas de modelação e simulação existentes não estão preparadas para a aplicação de técnicas de *model-checking* de forma transparente. Em particular, estas não escondem do utilizador os detalhes de instalação de um *model checker*, a exportação de um modelo e de propriedades num formato adequado, a chamada de execução de um *model checker* e finalmente a importação dos resultados produzidos (o veredicto verdadeiro/falso e a testemunha/contra-exemplo) de volta ao ambiente de modelação.

De forma a minimizar cada uma das limitações da utilização de técnicas de *model-checking* na análise de redes biológicas, são propostas três soluções diferentes no âmbito desta tese.

Primeiro, é proposta uma nova lógica temporal, denominada *Computation Tree Regular Logic* (CTRL), suficientemente expressiva para representar propriedades biológicas como a multi-estabilidade (tempo-ramificado) e as oscilações (tempo-linear). A nova lógica CTRL estende a lógica CTL usando expressões regulares e operadores de justiça, conseguindo um bom compromisso entre poder expressivo, usabilidade e complexidade de *model-checking*. Segundo, é proposto um sistema de padrões, baseado em perguntas frequentemente formuladas por quem estuda e modela redes biológicas. Cada padrão é uma pergunta que utiliza uma linguagem natural estruturada, sendo posteriormente traduzida, de forma automática, para formulas em lógica temporal. Terceiro, é proposta uma arquitectura baseada em serviços capaz de integrar ferramentas de modelação e simulação com ferramentas de *model-checking*, fornecendo um ambiente comum para a modelação e verificação de redes de regulação biológicas. Esta arquitectura é genérica e modular, permitindo a integração de diferentes *model checkers* através do uso de um sistema de *plugins* e permitindo a ligação de ferramentas de modelação e simulação diferentes.

Os métodos propostos foram implementados na nova versão da ferramenta Genetic Network Analyzer (GNA 7.0), através do desenvolvimento de um novo módulo de verificação. Este módulo fornece uma interface gráfica capaz de ajudar o utilizador na especificação de propriedades biológicas, bem como na visualização e interpretação dos resultados. De forma a demonstrar o uso e a aplicabilidade dos métodos desenvolvidos, o GNA 7.0 foi usado na validação de dois modelos biológicos complexos. O primeiro é um modelo de uma rede de reguladores que controlam a resposta à ausência de carbono na bactéria *Escherichia coli*. O segundo é um modelo de uma rede de activação do gene *FLR1*, codificador de um transportador de membrana celular, na resposta à presença do fungicida *mancozeb* no organismo eucariota *Saccharomyces cerevisiae*. Em ambos os modelos foi verificada a existência de inconsistências entre as predições do modelo e os dados experimentais disponíveis. As inconsistências identificadas deram origem a novas experiências laboratoriais e a revisões do modelo computacional.

Os métodos de verificação formal são ferramentas promissoras para o problema da escalabilidade da análise de redes de regulação biológicas. A implementação dos métodos propostos fornece ao modelador um ambiente único de integração que abre novas perspectivas, como é o caso da exploração de utilização de abordagens simbólicas para permitir a análise de modelos maiores e mais complexos, e ainda a definição de diagnósticos para a revisão semi-automática de modelos.

Palavras chave: biologia de sistemas, redes de regulação biológicas, simulação qualitativa, verificação formal, lógica temporal, *model checking*.

Résumé

L'étude de réseaux biologiques donne lieu à des modèles de plus en plus grands et détaillés. Bien que des modèles complets de la cellule ne soient pas encore d'actualité, des réseaux complexes qui interviennent dans certains processus cellulaires spécifiques ont déjà été modélisés en détail. L'étude de ces modèles par l'utilisation d'outils d'analyse et de simulation conduit à un grand nombre de prédictions, généralement sous forme d'évolutions temporelles de la concentration de plusieurs dizaines de composants moléculaires sous une variété de conditions physiologiques et génétiques. Cela soulève la question de savoir comment ces simulations peuvent nous aider à mieux comprendre la manière dont les mécanismes moléculaires contrôlent un processus cellulaire donné, et comment identifier les prédictions intéressantes de nouveaux phénomènes, qui peuvent ensuite être confrontées à des données expérimentales.

Les techniques de vérification formelle basées sur le *model checking* constituent une technologie puissante pour faire face à cette augmentation d'échelle et de complexité. L'idée de base du *model checking* est de spécifier des propriétés dynamiques intéressantes sous forme de déclarations en logique temporelle, et d'utiliser des algorithmes de *model checking* pour vérifier de façon automatique et efficace si les propriétés sont satisfaites ou non par le modèle. L'application directe des techniques de *model checking* est par contre dificile, pour plusieurs raisons.

Premièrement, nous verrons que le domaine de la biologie des systèmes demande l'analyse de quelques propriétés dynamiques du réseau qui ne sont pas facilement exprimables avec les logiques temporelles classiques, comme par example *Computation Tree Logic* (CTL) et *Linear Time Logic* (LTL). Par example, des questions sur la multistabilité sont importantes dans l'analyse de réseaux de régulation biologiques, mais sont difficiles (voire impossibles) à exprimer à l'aide de LTL. Aussi, CTL est capable de traiter le branchement temporel, ce qui est important pour traiter la multistabilité et d'autres propriétés de modèles non-déterministes, mais n'est pas capable de formuler des requêtes concernant les cycles dans une structure de Kripke. Ensuite, le difficulté de poser des questions pertinentes et intéressantes est essentielle dans la modélisation en général, mais c'est encore plus le cas dans le contexte de l'application des techniques de *model checking*, parce qu'il n'est pas facile pour des non-experts de formuler des requêtes en logique temporelle. Enfin, la plupart des modèles existants et des outils de simulation ne sont pas capables d'appliquer des techniques de *model checking* d'une manière transparente. En particulier, ils ne cachent pas à l'utilisateur les détails techniques de l'installation du *model checker*, l'export du modèle et de la requête dans un format adapté, l'appel du *model checker*, et l'import des résultats produits par le *model checker* (le verdict vrai/faux et le témoin/contre-exemple).

Dans le contexte de cette thèse, nous proposons trois approches différentes pour aborder chacune des questions. Premièrement, nous proposons une nouvelle logique temporelle, appelée *Computation Tree Regular Logic* (CTRL), qui est suffisamment puissante pour cap-

turer les propriétés biologiques de multistabilité (branchement temporel) et des oscillations. CTRL étend CTL avec des expressions régulières et des opérateurs d'équité, en cherchant un bon compromis entre l'expressivité de la logique, la la facilité d'utilisation et la complexité des algorithmes de *model checking*. Deuxièmement, nous proposons un système de patterns nouveau, basé sur les questions fréquemment posées par les modélisateurs en biologie des systèmes. Chaque pattern est une requête de haut niveau, formulée en langage naturel structuré au lieu d'une formule de logique temporelle, qui peut être traduite automatiquement dans des formules en logique temporelle. Troisièmement, nous proposons une architecture orientée services capable d'intégrer des outils de modélisation et de simulation avec des outils existants de *model checking*, de façon à fournir un environnement commun pour la modélisation et la vérification de réseaux de régulation biologiques. Cette architecture est générique et modulaire, capable d'intégrer des *model checkers* différents grâce à l'utilisation d'un système de plugins, et permettant en principe aussi l'intégration de différents outils de modélisation et simulation.

Les méthodes qui sont proposées ont été mises en ouvre dans une nouvelle version de l'outil Genetic Network Analyzer (GNA 7.0), à travers du développement d'un nouveaux module de vérification. Ce module fournit une interface utilisateur graphique capable d'aider l'utilisateur avec la spécification des propriétés biologiques, ainsi que la visualisation et l'interprétation des résultats de la vérification. Afin d'illustrer l'utilisation et l'applicabilité des méthodes développées, GNA 7.0 a été utilisé pour la validation de deux modèles biologiques complexes. Le premier est un modèle du réseau des régulateurs globaux qui contrôle la réponse à l'épuisement de sources de carbone chez la bactérie *Escherichia coli*. Le deuxième est un modèle de l'activation du 12-H^+ anti-porteur des drogues, codé par le gène *FLR1*, dans la présence du fongicide *mancozeb* chez l'eucaryote *Saccharomyces cerevisiae*. Ces deux modèles ont été analysées en confrontant les prédictions du modèle avec des données expérimentales disponibles. Les incohérences trouvées ont mené à des expériences complémentaires et/ou à des révisions du modèle.

Les méthodes de vérification formelle sont des outils prometteurs pour l'analyse de réseaux de régulation biologiques de plus en plus grands. La mise en oeuvre des approches proposées dans un environnement unique intégré permet à l'utilisateur une méthode automatique pour l'analyse de ces réseaux complexes. Cette thèse contribue à enlever des obstacles pour l'utilisation de la technologie de vérification formelle en biologie, et ouvre une voie à de nouvelles perspectives comme l'exploration d'approches symboliques pour continuer à augmenter la taille et la complexité de modèles, et la définition de diagnostiques pour la révision semi-automatique de modèles.

Mots-clés: biologie des systèmes, réseaux de régulation biologique, simulation qualitative, vérification formelle, logique temporelle, *model checking*.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary of Abbreviations

CTMC	-	Continuous Time Markov Chain
FSTS	-	Finite State Transition System
AP	-	Atomic proposition
BDD	-	Binary Decision Diagram
BES	-	Boolean Equation System
MES	-	Modular Equation System
RES	-	Regular Equation System
PNF	-	Positive Normal Form
CTL	-	Computation Tree Logic
LTL	-	Linear Time Logic
CTRL	-	Computation Tree Regular Logic
CSL	-	Continuous Stochastic Logic
HML	-	Hennessy-Milner Logic
HMLR	-	Hennessy-Milner Logic with Recursion
RCTL	-	Regular CTL
PDL	-	Propositional Dynamic Logic
ETL	-	Extended Temporal Logic
GNA	-	Genetic Network Analyzer
SMV	-	Symbolic Model Verifier
NUSMV	-	New Symbolic Model Verifier
GINSIM	-	Gene Interaction Network simulation
BIOCHAM	-	Biochemical Abstract Machine
CADP	-	Construction and Analysis of Distributed Processes

Part I

Background

Chapter 1

Introduction

Cellular interaction networks consist of genes, proteins, small molecules, and their mutual interactions, which are involved in the control of cellular functions. Most of these networks are large and complex, thus defying our capacity to understand how the dynamic behavior of the cell emerges from the structure of interactions. This has led to the development of increasingly large and detailed models [77], consisting of dozens or even hundreds of variables describing the molecular species involved in a variety of intracellular processes [34, 33, 58, 81, 94, 120]. The study of these models by means of analysis and simulation tools leads to large amounts of predictions, typically time-courses of the concentration of several dozens of molecular components in a variety of physiological conditions and genetic backgrounds. This raises the question how to make sense of these simulations, that is, how to obtain an understanding of the way in which particular molecular mechanisms control the cellular process under study, and how to identify interesting predictions of novel phenomena that can be confronted with experimental data.

For instance, Figure 1.1 presents a interaction network between different functional modules involved in the response of *Saccharomyces cerevisiae* to osmotic shock. This complex network may be used to answer questions like the following "Is the basal glycerol production level combined with rapid closure of Fps1 sufficient to explain an initial glycerol accumulation after osmotic shock?". Another question that arises is how to encode these biological questions in a non-ambiguous formal manner. In addition, it is clear that verifying the existence of a particular behavior against all the predictions generated by simulation tools, quickly becomes a challenging task that cannot be manually performed.

Methods from the field of formal verification provide a promising way to deal with the analysis of large and complex models of cellular interaction networks [56]. These methods proceed by an exploration of all possible behaviors of the system, following two main approaches: logic inference, based on the use of axioms and proof rules [99], and model checking, based on an automatic and exhaustive search of the state space [40]. Model checking is the most used approach, and its basic idea is to specify dynamical properties of interest as statements in

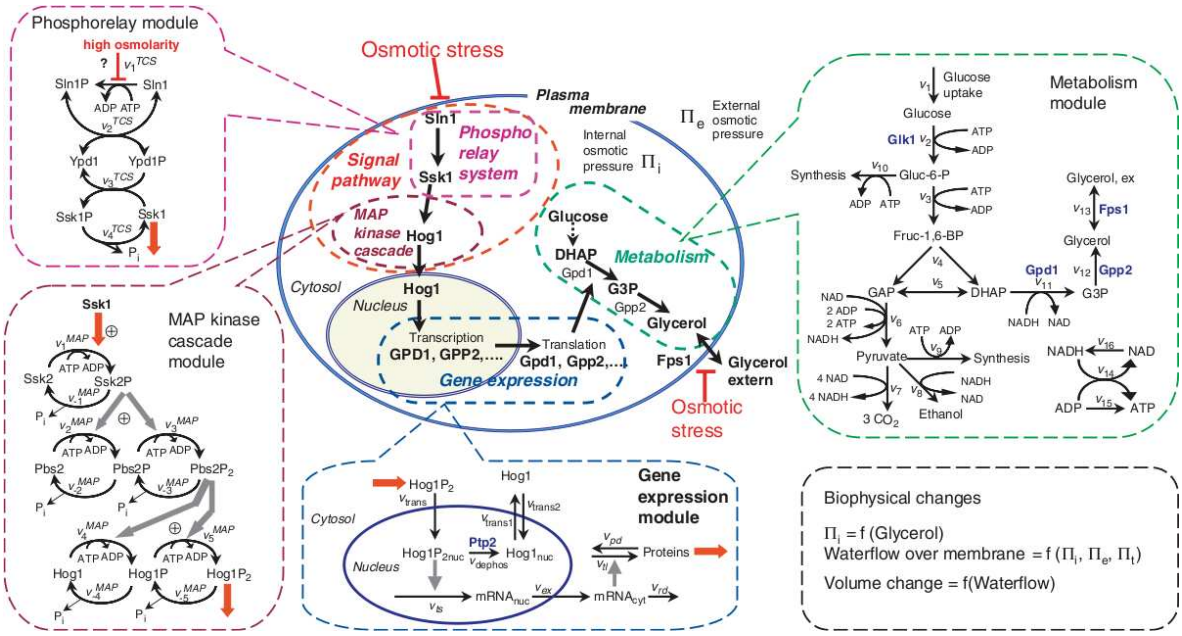


Figure 1.1: Overview of the response of *Saccharomyces cerevisiae* to hyperosmotic stress, divided into several functional modules defined by level of biological organization [81].

temporal logic, and to use model checking algorithms to automatically and efficiently verify whether the properties are satisfied or not by the model [40]. In recent years, several examples of the application of model checking to the analysis of biological regulatory networks have been published in the literature (*e.g.*, [4, 8, 9, 14, 15, 18, 21, 26, 29, 64, 57, 123]).

1.1 Problem

Three major problems exist regarding the use of formal verification techniques by modelers in systems biology. First, the study of biological systems has recently shown that classical temporal logics are not expressive enough to describe accurately the dynamical behavior of some biological properties needing both multistability (branching-time) and oscillations (linear-time). Second, the formulation of biological questions in temporal logic and the interpretation of the verification results is far from obvious, especially for non-expert users who are not used to this kind of reasoning. Third, most of the existing modeling and simulation tools are not capable of applying model checking techniques in a transparent way. In particular, they do not hide from the user the intricates of temporal logic formulas and details of verification procedures, and they do not provide good graphical user interfaces for the presentation of the verification results.

Non-expert users, especially modelers in systems biology are thus impaired at several levels to use model checking techniques, hindering the wide-spread use of these techniques in

biological modeling.

1.2 Approach

In order to address these issues, four specific research lines are proposed in the context of this PhD thesis.

To tackle the first issue, the temporal logic expressiveness problem, we propose to devise a new temporal logic powerful enough to capture the biological properties of multistability (branching-time) and oscillations (linear-time). This temporal logic should have a small number of operators, and a succinct and intuitive user-friendly syntax for non-expert users. Also, it should be state-based in order to be directly interpreted on Kripke structures (see Section 2.1 for formal definition), which provide a general description of dynamic systems. Also, this temporal logic should have a reasonable model checking complexity.

There is however, a syntax and semantics associated to all temporal logics that needs to be understood by users. Modelers in systems biology are not expected to know it in order to formulate biological questions. For this issue, we propose to devise a set of query pattern templates, based on a review of frequently-asked questions by modelers, and to automatically translate these templates into temporal logic formulas.

To solve the third issue, we propose an implementation of a service-oriented architecture, connecting the modeling and simulation tools to model checking servers. This architecture should abstract the users from all the verification details and provide a graphical user interface for the presentation of the verification results.

Finally, in order to validate the proposed approach and to illustrate its applicability in a real case scenario, we propose the analysis of two large and complex biological models of genetic regulatory networks whose predictions cannot be manually verified.

1.3 Contributions

We developed several contributions in the context of this thesis.

First, we defined a new temporal logic, named Computation Tree Regular Logic (Ctrl), adapted to the specification of biological concepts. Most temporal logics are either branching-time or linear-time, allowing the specification of either multistability or oscillations, respectively. The use of temporal logics of higher expressive power, like propositional μ-calculus [84], permits the specification of both these biological properties. However, as the expressive power of a temporal logic increases, so does the complexity of its semantics, and therefore the difficulty of specifying a property. Ctrl extends Ctl [39] with regular expressions and fairness operators, achieving a good compromise between expressiveness, user-friendliness and complexity of model checking. A model checker for Ctrl was obtained through the implementation of a translation from Ctrl to Hennessy-Milner Logic with Recursion and the reuse of

the verification engine CADP [61]. This work has resulted in a publication in the *Automated Technology for Verification and Analysis* conference [102] and is under revision for a computer science journal.

Second, we developed a set of patterns capable of capturing frequently-asked biological questions, by formulating them in the form of query templates in structured natural language. We started by performing a careful analysis of previous modeling studies and applications of formal verification in systems biology, and we realized that a set of questions emerged as the most frequently-asked by modelers. We are able to provide the user this set of patterns together with the corresponding variants, corresponding to dynamic behaviors like reachability, sequences or invariance of events. The patterns can then be automatically translated into temporal logic formulas without the user intervention. This work was published in a special issue of *Bioinformatics* [109].

Third, we developed a service-oriented architecture which connects the modeling and simulation clients to a model-checking server *via* web-service technology. This connection enables users to access formal verification technology through a single environment through the modeling and simulation tools. It hides from the user all the formal verification technical details like the export of the model and the query into a model-checker specific format, the call of the model checker, and the import and interpretation of its results. This connection was implemented and is fully operational through the Genetic Network Analyzer (GNA 7.0) modeling and simulation tool which connects to the implemented formal verification servers at INRIA Grenoble Rhône-Alpes, supporting the NUSMV and CADP model checkers. This work has resulted in a publication in *BMC Bioinformatics* [108].

Finally, the whole infrastructure was validated by the analysis of two biological systems in two model organisms, the bacterium *Escherichia coli* and the eukaryote *Saccharomyces cerevisiae*. The first model describes a complex genetic regulatory network involved in the carbon starvation response in *E. coli*, extending a previous model [118] with additional regulators to better account for the control of DNA supercoiling during growth transitions of the bacteria. This work has resulted in the publication [109]. The second model describes a genetic regulatory network involved in the stress response to the drug mancozeb in *S. cerevisiae*, by taking into account the cascade of transcription factors that activate the gene FLR1, responsible for exporting the drug out of the cell. Preliminary results were published in [106], while a complete description of the model was submitted for publication in *BMC Systems Biology* [107].

1.4 Thesis outline

This thesis is organized as follows:

Chapter 2 introduces fundamental notions of finite state transition systems, temporal logic and the basic model checking problem with a running example. Also, applications of model checking to the field of systems biology are shown, stressing their problems and limitations.

Chapter 3 describes the first contribution, the CTRL language, including its syntax, semantics and expressiveness. It is also presented the translation steps to Hennessy-Milner logic with recursion (HMLR) [92] and a description of the connection to the EVALUATOR model checker of the CADP toolbox [61].

Chapter 4 describes the second contribution, a pattern system in the form of query templates in structured natural language, capable of capturing frequently-asked questions posed by modelers in systems biology. It is also presented the translations of the patterns to several temporal logics.

Chapter 5 describes the third contribution, the implementation of a framework that integrates modeling and simulation tools with model checkers through a service-oriented architecture. It is also described the necessary translation steps for the model and the properties from the modeling and simulation tools towards the model checkers and vice-versa.

Chapter 6 presents a validation of the developed system by applying it to the verification of the biological models of two organisms, the bacterium *E. coli* and the eukaryote *S. cerevisiae*.

Chapter 7 summarizes the developed contributions and places them in context of the current use of formal verification techniques in the field of systems biology.

Appendix A presents the logic proofs that support the semantic definition of the temporal logic CTRL, as well as the necessary translation steps to transform a CTRL temporal logic formula into a HMLR equation block.

Appendix B shows some screenshots of the GNA usage to support the analysis of two genetic regulatory networks in Chapter 6.

Chapter 2

Related work

The application of methods from the field of formal verification for the analysis of biological complex models, putted forward the necessity to understand concepts coming from both fields.

This chapter starts by introducing the basic concepts and data structures that are used to represent the possible behaviors of a dynamical system. Next, a brief overview is given over some of the different types of existing temporal logics necessary for the specification of properties in a system. It also presents the model checking problem, which consists in determining if a given system satisfies a given set of properties.

Some of the recent applications of model checking techniques in systems biology are also presented in Section 2.4. These applications concern different types of biological systems using different mathematical formalisms to represent the model. However, they all share the necessity to represent the behaviors of the system as a FSTS and to use temporal logic as means to specify dynamic properties.

Finally, despite the existence of successful applications of model checking techniques for querying biological systems, the usage of these techniques is not natural for most users, especially those coming from the systems biology community. Section 2.5 presents some problems and limitations that currently hamper the natural use of these techniques.

2.1 Finite state transition systems

In the context of this thesis, the problem of determining whether the dynamical behavior of a biological model satisfy some properties, must be translated into a model checking problem (see Section 2.3 for formal definition).

The application of formal verification techniques to computational systems biology corresponds to the same methodology as the one developed for hardware and software verification. In order to reuse it, one must transform the systems biology problem into a hardware/software problem. The most used formal verification method is model checking, typically representing the behavior of a discrete system or a continuous one that been discretized under a suit-

able abstraction criterion, through a graph structure (implicit or explicit), where the nodes represent the states of the system and the edges the possible transitions between the states.

This graph will contain information about the system, where each state of the graph represents the state of the system at a given instant. This specific information contained in a state is often called *atomic proposition*. The atomic propositions considered in this thesis contain information about the concentration bounds of the variables, the trend of the variables in that state (increasing, decreasing or steady), among other things.

There are two main approaches to represent atomic propositions in model checking: using *Kripke structures*, introduced by Saul Kripke in 1963 [86], where this information is annotated on the nodes by means of atomic propositions, or by using *labeled transition systems* [113], where this information is annotated on the transition arcs, by means of labels called *actions*.

Definition 1 (Kripke Structure) *A Kripke structure K is tuple $\langle S, S_0, AP, L, TR\rangle$ where S is the set of states, $S_0 \subseteq S$ is the set of initial states, AP is the set of atomic propositions, $L : S \rightarrow 2^{AP}$ is the state labeling function which labels each state with a set of atomic propositions true in that state, $TR \subseteq S \times S$ is the transition relation which is assumed to be total (i.e., for each state $s \in S$ there exists a state $s' \in S$ such that $(s, s') \in T$).*

A state s in the Kripke structure is characterized by its label, which contain the set of atomic propositions $Q_s \subseteq AP$ that are valid in that state.

Definition 2 (Labeled Transition System) *A labeled transition system LTS is tuple $\langle S, S_0, Act, \rightarrow\rangle$ where S is the set of states, $S_0 \subseteq S$ is the set of initial states, Act is a set of actions, $\rightarrow \subseteq S \times Act \times S$ is the transition relation. A transition $(s, a, s') \in \rightarrow$ if the system can move from s to s' by performing action a, also notated $s \xrightarrow{a} s'$.*

Contrary to the Kripke structure where each state is labeled with a set of atomic propositions, in a labeled transition system the transitions are labeled with single actions.

Definition 3 (Computation tree) *A computation tree is an acyclic (possibly infinite) graph of nodes and edges, where each node represents a single computation state and each edge represents a transition to the next possible computation state.*

One can imagine the unwinding of the state transition system in Figure 2.1 into a data structure called *computation tree* (Figure 2.2) where the level of the tree represents time and a given state on the computation tree represents the current state, having one or more successors representing the possible future states.

2.2 Temporal logic

Temporal logic is a formalism for describing sequences of transitions between states in a reactive system. In classical temporal logics the notion of time is not mentioned explicitly [40].

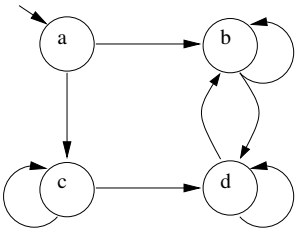


Figure 2.1: State transition system.

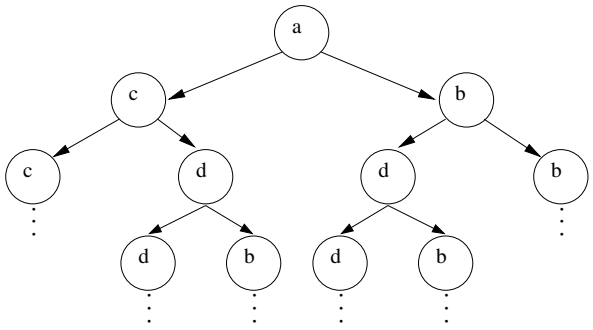


Figure 2.2: Computation tree.

Instead, the notion of time-order is introduced through the use of *temporal operators* to describe statements like some designated state is *eventually* reached, or that an error state is *never* reached.

Several temporal logics have been defined since the 80s, differing in many characteristics. All of them however, can be characterized based on two main characteristics. The first characteristic separates *linear time* logics, where each execution has one single possible future, from the *branching time* logics, where at each instant a choice can be made between alternate courses representing multiple possible futures. The second characteristic separates the logics that are *state based* from those that are *action based*, in other words, those that associate atomic propositions to states from those that associate actions to transitions.

The most used branching-time state-based logic is CTL [39] and due to its simplicity it will be used to explain the basic syntax and semantic of temporal logic formulas. Other temporal logics will be discussed as well, differing on their expressive power and their way of representing a given property.

2.2.1 Computation Tree Logic (CTL)

CTL formulas are composed of the standard propositional operators together with temporal operators and path quantifiers [39]. These formulas describe properties of computation trees, which are formed by picking an initial state in a Kripke structure and unwinding the structure into an infinite tree with the designated state at the root (Figure 2.2). This resulting computation tree shows all the possible executions starting from that designated state. The notion of computation path refers to a (not necessarily infinite) set of sequentially connected states along the computation tree.

There are two types of formulas in CTL: state formulas (which are true in a specific state) and path formulas (which are true in a specific path) [40]. In order to describe the branching structure in the computation tree, there are two path quantifiers (Table 2.1). The existential (**E**) and the universal (**A**) path quantifiers denote that a given path formula f at state s, should hold for some computation path or for all computation paths, respectively.

In order to describe properties of a path along the tree, temporal operators are used (Table 2.1). Given a state formula ψ and a current path starting at state s, the formula $\mathsf{X}\ \psi$ (**X** for next) is valid if the next state along the path satisfies the property ψ. The formula $\mathsf{F}\ \psi$ (**F** for future) is valid if some future state along the path satisfies the property ψ. The formula $\mathsf{G}\ \psi$ (**G** for globally) is valid if every state along the path satisfies the property ψ. The formula $\phi\ \mathsf{U}\ \psi$ (**U** for until) is valid if some state along the path satisfies ψ and every preceding state on that path satisfies ϕ.

$\mathsf{E}\ (f)$	"Exists"	f has to hold for some computation path
$\mathsf{A}\ (f)$	"For all"	f has to hold for all computation paths
$\mathsf{X}\ \psi$	"Next"	ψ has to hold at the next state
$\mathsf{F}\ \psi$	"Future"	ψ eventually has to hold on some future state
$\mathsf{G}\ \psi$	"Globally"	ψ has to hold at the entire subsequent path
$\phi\ \mathsf{U}\ \psi$	"Until"	ϕ has to hold in all states up until ψ holds (this implies that ψ *will be* verified in the future)

Table 2.1: CTL path quantifiers and temporal operators.

2.2.1.1 CTL syntax and semantics

By combining the previously defined path quantifiers with the temporal operators, in such a way that each temporal operator is always immediately preceded by a path quantifier, we obtain the following list of CTL operators defining the syntax for CTL formulas:

$$\begin{aligned}\phi, \psi \quad ::= \quad & p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \\ & \mid \mathsf{EX}\ \phi \mid \mathsf{AX}\ \phi \mid \mathsf{EF}\ \phi \mid \mathsf{AF}\ \phi \mid \mathsf{EG}\ \phi \mid \mathsf{AG}\ \phi \mid \mathsf{E}[\phi\ \mathsf{U}\ \psi] \mid \mathsf{A}[\phi\ \mathsf{U}\ \psi]\end{aligned}$$

where p ranges over a (finite) set AP of atomic propositions.

We define the semantics of CTL with respect to a Kripke structure M (Section 2.1). We use π_i to denote the ith state of a path π. The interval going from the ith state of a path π to the jth state inclusively (where i $\leq$ j) is noted $\pi_{i,j}$. An interval $\pi_{0,i}$ is called prefix of π. For each state $s \in S$, $Path(s)$ denotes the set of all paths going out of s, *i.e.*, the paths π such that $\pi_0 = s$. The semantics of the CTL language is described in Table 2.2.

A state s of M satisfies the atomic proposition p (notation $M, s \vDash p$) if and only if $p \subset Labels(s)$, where $labels(s)$ denotes the set of all atomic propositions belonging to state s. The semantics of boolean operators is defined in the standard way. A state satisfies the formula $\mathsf{EX}\ \phi$ if and only if it has at least a successor satisfying ϕ. A state satisfies the formula $\mathsf{AX}\ \phi$ if and only if all its successors satisfy ϕ. A state satisfies the formula $\mathsf{EF}\ \phi$ if and only if it has an outgoing path leading to a state satisfying ϕ. A state satisfies the formula $\mathsf{AF}\ \phi$ if and only if all its outgoing paths lead to states satisfying ϕ. A state satisfies the formula $\mathsf{EG}\ \phi$ if and only if it has an outgoing path with all its states satisfying ϕ. A state satisfies

$M, s \vDash p$	$\Leftrightarrow$	$p \in L(s)$
$M, s \vDash \neg\phi$	$\Leftrightarrow$	$M, s \nvDash \phi$
$M, s \vDash \phi_1 \vee \phi_2$	$\Leftrightarrow$	$M, s \vDash \phi_1$ or $M, s \vDash \phi_2$
$M, s \vDash \phi_1 \wedge \phi_2$	$\Leftrightarrow$	$M, s \vDash \phi_1$ and $M, s \vDash \phi_2$
$M, s \vDash \mathsf{EX}\ \phi$	$\Leftrightarrow$	$\exists_{\pi \in Path(s)}.M, \pi_1 \vDash \phi$
$M, s \vDash \mathsf{AX}\ \phi$	$\Leftrightarrow$	$\forall_{\pi \in Path(s)}.M, \pi_1 \vDash \phi$
$M, s \vDash \mathsf{EF}\ \phi$	$\Leftrightarrow$	$\exists_{\pi \in Path(s)}.\exists_{i \geq 0}.M, \pi_i \vDash \phi$
$M, s \vDash \mathsf{AF}\ \phi$	$\Leftrightarrow$	$\forall_{\pi \in Path(s)}.\exists_{i \geq 0}.M, \pi_i \vDash \phi$
$M, s \vDash \mathsf{EG}\ \phi$	$\Leftrightarrow$	$\exists_{\pi \in Path(s)}.\forall_{i \geq 0}.M, \pi_i \vDash \phi$
$M, s \vDash \mathsf{AG}\ \phi$	$\Leftrightarrow$	$\forall_{\pi \in Path(s)}.\forall_{i \geq 0}.M, \pi_i \vDash \phi$
$M, s \vDash \mathsf{E}[\phi_1\ \mathsf{U}\ \phi_2]$	$\Leftrightarrow$	$\exists_{\pi \in Path(s)}.\exists_{i \geq 0}.\forall_{0 \leq j < i}.M, \pi_j \vDash \phi_1 \wedge M, \pi_i \vDash \phi_2$
$M, s \vDash \mathsf{A}[\phi_1\ \mathsf{U}\ \phi_2]$	$\Leftrightarrow$	$\forall_{\pi \in Path(s)}.\exists_{i \geq 0}.\forall_{0 \leq j < i}.M, \pi_j \vDash \phi_1 \wedge M, \pi_i \vDash \phi_2$

Table 2.2: Semantics of CTL [40].

the formula $\mathsf{AG}\ \phi$ if and only if all its outgoing paths have all their states satisfying ϕ. A state satisfies the formula $\mathsf{E}[\phi_1\ \mathsf{U}\ \phi_2]$ if and only if it has an outgoing path leading to a state satisfying ϕ_2 after zero or more states satisfying ϕ_1. A state satisfies the formula $A[\phi_1\ \mathsf{U}\ \phi_2]$ if and only if all its outgoing paths lead to states ϕ_2 after zero or more states satisfying ϕ_1.

The four operators that are used most widely are illustrated in Figure 2.3. These operators are easy to understand in terms of the computation tree obtained by unfolding the Kripke structure, having the state s as its root.

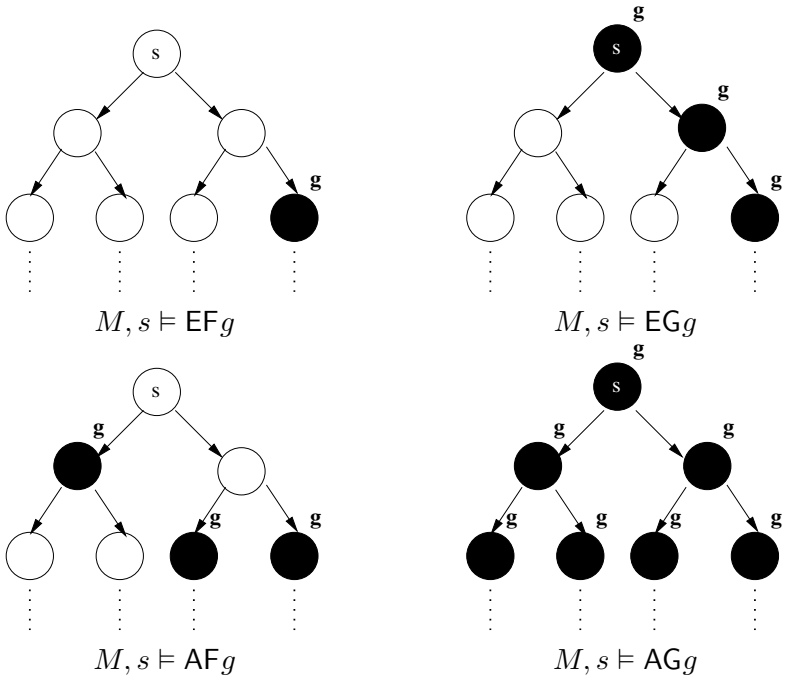


Figure 2.3: Most widely used CTL operators [40].

By looking at the semantics of each operator it can be easily seen that the path quantifiers

are dual between themselves, as well as some of the temporal operators. The existing dualities and some equivalences are presented in Table 2.3.

$\mathsf{A}\ \phi$	$\equiv$	$\neg\mathsf{E}\ (\neg\phi)$
$\mathsf{G}\ \phi$	$\equiv$	$\neg\mathsf{F}\ (\neg\phi)$
$\mathsf{F}\ \phi$	$\equiv$	$\mathsf{true}\ \mathsf{U}\ \phi$
$\mathsf{AX}\ \phi$	$\equiv$	$\neg\mathsf{EX}\ (\neg\phi)$
$\mathsf{EF}\ \phi$	$\equiv$	$E[\mathsf{true}\ \mathsf{U}\ \phi]$
$\mathsf{AG}\ \phi$	$\equiv$	$\neg\mathsf{EF}\ (\neg\phi)$
$\mathsf{AF}\ \phi$	$\equiv$	$\neg\mathsf{EG}\ (\neg\phi)$
$\mathsf{A}[\phi\ \mathsf{U}\ \psi]$	$\equiv$	$\neg\mathsf{E}[(\neg\psi)\ \mathsf{U}\ (\neg\phi \wedge \neg\psi)] \wedge \neg\mathsf{EG}\ (\neg\psi)$

Table 2.3: Dualities and equivalences in CTL.

The use of CTL operators is illustrated bellow for the specification of biological properties. For instance, the property "gene g is never expressed" corresponds to the following CTL formula, where ϕ_g denotes the expression of gene g:

$$\mathsf{AG}\ (\phi_g = 0)$$

For a more complex example, the property "gene g is eventually expressed, and is necessarily preceded over the whole duration of the experiment by a concentration larger than 0.9 μM of the transcription factor P" corresponds to the following CTL formula, where ϕ_g denotes the expression of gene g:

$$\mathsf{EF}\ (\phi_g > 0) \wedge \mathsf{AG}\ (\neg([P] > 0.9\mu M) \Rightarrow \mathsf{AG}\ (\phi_g = 0))$$

2.2.2 Computation Tree Logic* (CTL*)

Starting from the syntax and semantics of the CTL language defined above, the CTL* language can be introduced by dropping the syntactic restriction stating that each temporal operator must be preceded by a path quantifier, resulting in the following syntax:

$$\phi, \psi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \mathsf{X}\ \phi \mid \mathsf{F}\ \phi \mid \mathsf{G}\ \phi \mid \phi\ \mathsf{U}\ \psi \mid \mathsf{E}\ \phi \mid \mathsf{A}\ \phi$$

where p ranges over a (finite) set AP of atomic propositions.

This means that the CTL* language can not only express the same formulas as CTL, but also express all the formulas composed by two or more consecutive temporal operators without having to precede each one of them with a path quantifier. CTL* is therefore more expressive than CTL [38]. Examples of CTL* formulas that are not expressible in CTL are the following:

$$\mathsf{E}\ (\mathsf{FG}\ p)$$

and

$$\mathsf{A}\ (\mathsf{FG}\ p) \vee \mathsf{AG}\ (\mathsf{EF}\ p)$$

2.2.3 Linear Time Logic (LTL)

The two previously described logics were state-based branching-time logics. LTL [40] is a linear-time state-based logic, less expressive than CTL* [51, 90], consisting on the suppression of the path quantifiers from its formulas, resulting in the following syntax:

$$\phi, \psi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \mathsf{X}\ \phi \mid \mathsf{F}\ \phi \mid \mathsf{G}\ \phi \mid \phi\ \mathsf{U}\ \psi$$

where p ranges over a (finite) set AP of atomic propositions.

The LTL formulas are in the form $\mathsf{A}\ (f)$, denoting that all the executions must satisfy formula f, where f is prevented from using path quantifiers to combine with the temporal operators. Examples of such formulas are the following:

$$\mathsf{FG}\ p$$

and

$$\mathsf{F}\ (p \Rightarrow \mathsf{G}\ p)$$

Although CTL and LTL are two subsets of CTL*, neither of them totally includes the other, and therefore their expressive power cannot be compared [40].

2.2.4 Modal μ-calculus

The μ-calculus is a modal logic using greatest and least fixpoint operators. It was originally introduced by Kozen [84], to be interpreted on Kripke structures with labeled transitions, and by Kupferman *et al.* [87], to be interpreted on classical Kripke structures.

μ-calculus formulas are composed of two types of operators: the least (μ) and greatest (ν) *fixed points*, and the *modal operators* possibility ($\Diamond$) and necessity ($\Box$). Intuitively, least fixpoints correspond to eventualities or finite recursions of a formula, while greatest fixpoints correspond to properties that should hold forever or infinite recursions of a formula. For instance, given a state and a path starting from that state, the fact that a property p holds for some state or for all states of the path is expressed using a least (μ) or a greatest (ν) fixed point, respectively. Modal operators are used to specify that, given a state, a property p possibly ($\Diamond\ p$) or necessarily ($\Box\ p$) holds on the next state of some or all of its outgoing paths.

Modal μ-calculus formulas are defined over a set of actions $a \in \mathcal{A}$ and a set of variables

$X \in \mathcal{X}$, and are constructed according to the following syntax:

$$\phi, \psi ::= \mathsf{true} \mid \mathsf{false} \mid p \mid \neg p \mid \phi \wedge \psi \mid \phi \vee \psi \mid [a]\phi \mid \langle a \rangle \phi \mid X \mid \mu X.\phi \mid \nu X.\phi$$

where p ranges over a (finite) set AP of atomic propositions.

To the biological property "gene g is never expressed", previously specified in CTL, where ϕ_g denotes the expression of the gene g, corresponds the following μ-calculus formula:

$$\nu X.((\phi_g = 0) \wedge [\mathsf{true}]X)$$

To the second biological property stating that "gene g is eventually expressed, and is necessarily preceded over the whole duration of the experiment by a concentration larger than 0.9μM of the transcription factor P", corresponds the following μ-calculus formula:

$$\mu X.((\phi_g > 0) \vee \langle \mathsf{true} \rangle X) \wedge \nu Y.((\neg([P] > 0.9\mu M) \Rightarrow \nu Z.((\phi_g = 0) \wedge [\mathsf{true}]Z)) \wedge [\mathsf{true}]Y)$$

μ-calculus is more expressive than CTL or even CTL* [43]. Some of the properties presented in Chapter 6 cannot be expressed in CTL and need a temporal logic as expressive as μ-calculus. But, to a more expressive temporal logic typically corresponds to a more complex syntax, which in practice means that even the simplest properties are difficult to specify, especially for a non-expert user.

2.2.5 Hennessy-Milner Logic with Recursion (HMLR)

Hennessy-Milner Logic (HML) is a simple modal logic introduced by Hennessy and Milner in [71], considered a standard for modal logics interpreted on labeled transition systems. It has a limited expressiveness, but its presentation is of interest because it defines the basis for the Hennessy-Milner Logic with recursion (HMLR) that will be used in CTRL.

HML is defined over a set of actions $a \in \mathcal{A}$, and the formulas follow the following syntax:

$$\varphi ::= \mathsf{true} \mid \mathsf{false} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid [a]\varphi \mid \langle a \rangle \varphi$$

Given a LTS $T = \langle S, \mathcal{A}, \rightarrow \rangle$, the inductive definition of when a state $s \in S$ satisfies the formula φ is given by the following rules:

$s \vDash \mathsf{true}$		$s \nvDash \mathsf{false}$
$s \vDash \varphi_1 \wedge \varphi_2$	iff	$s \vDash \varphi_1$ and $s \vDash \varphi_2$
$s \vDash \varphi_1 \vee \varphi_2$	iff	$s \vDash \varphi_1$ or $s \vDash \varphi_2$
$s \vDash [a]\varphi$	iff	for all t with $s \xrightarrow{a} t$, $t \vDash \varphi$
$s \vDash \langle a \rangle \varphi$	iff	there is t with $s \xrightarrow{a} t$ and $t \vDash \varphi$

The branching time modalities $[a]$ and $\langle a \rangle$ relate a state to its a-successors (Figure 2.4).

While $[a]$ holds for a state if *all* its a-successors satisfy formula φ, $\langle a \rangle$ holds if an a-successor satisfying formula φ *exists*.

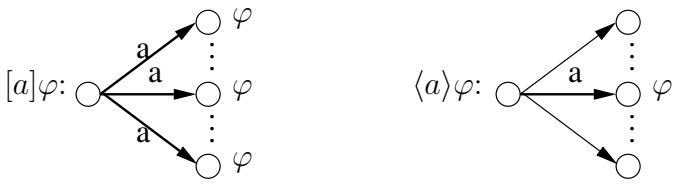


Figure 2.4: Illustration of branching-time modalities.

HmlR [92] replaces the fixpoint operators of the modal μ-calculus by blocks of recursive equations, where all equations belonging to a given block have the same fixpoint sign (μ or ν). Each equation is composed on the left by a propositional variable and on the right by an Hml formula.

Definition 4 (Fixed-point equation block) *A fixed-point equation block is a set of equations sharing the same fixed-point sign where the unguarded occurrences of variables occurring on the right-hand side of the equations are bounded by equations of the same block if the fixed-point sign is kept, or by equations of another block if the fixed-point sign is changed.*

To the biological property "gene g is never expressed", previously specified in Ctl, where ϕ_g denotes the expression of the gene g, corresponds the following HmlR equation block:

$$\langle X_1, \{X_1 \stackrel{\nu}{=} X_2 \wedge X_3, X_2 \stackrel{\nu}{=} [\mathsf{true}]X_1, X_3 \stackrel{\nu}{=} \langle(\phi_g = 0)\rangle\mathsf{true}\}\rangle$$

where X_1 is the main variable and must be bound in the first block of the equation block. The response to the biological property is obtained by interpreting the equation block on a Lts and obtaining the value of the variable X_1.

To the second biological property stating that "gene g is eventually expressed, and is necessarily preceded over the whole duration of the experiment by a concentration larger than 0.9μM of the transcription factor P", corresponds the following HmlR specification containing five equation blocks:

$$\begin{aligned}
&\langle X_1, \{X_1 \stackrel{\mu}{=} X_2 \wedge X_5\},\\
&\quad \{X_2 \stackrel{\mu}{=} X_3 \vee X_4, X_3 \stackrel{\mu}{=} \langle\mathsf{true}\rangle X_2, X_4 \stackrel{\mu}{=} \langle(\phi_g > 0)\rangle\mathsf{true}\},\\
&\quad \{X_5 \stackrel{\nu}{=} X_6 \wedge X_7, X_6 \stackrel{\nu}{=} [\mathsf{true}]X_5\},\\
&\quad \{X_7 \stackrel{\mu}{=} X_8 \vee X_9, X_8 \stackrel{\mu}{=} \langle(P > 0.9\mu M)\rangle\mathsf{true}\},\\
&\quad \{X_9 \stackrel{\nu}{=} X_{10} \wedge X_{11}, X_{10} \stackrel{\nu}{=} [\mathsf{true}]X_9, X_{11} \stackrel{\nu}{=} \langle(\phi_g = 0)\rangle\mathsf{true}\}\rangle
\end{aligned}$$

The Ctl formula $\mathsf{AG}(\varphi \Rightarrow \mathsf{EF}\psi)$ is represented in HmlR by two equation blocks: $\{X_1 \stackrel{\nu}{=} [\mathsf{true}]X_1 \wedge X_2\}$ and $\{X_2 \stackrel{\mu}{=} \langle\neg\varphi\rangle\mathsf{true} \vee X_3, X_3 \stackrel{\mu}{=} \langle\psi\rangle\mathsf{true} \vee \langle\mathsf{true}\rangle X_3\}$, where the main variable X_1 represents the property.

HmlR is as expressive as the modal μ-calculus and will be used later in this thesis as a target language for the translation process described in Chapter 3, to be interpreted by the Evaluator model checker.

2.3 Model checking

In this section we describe model checking procedures. Throughout the years several approaches and improvements were developed, like for example the use of symbolic model checking using Binary Decision Diagrams (Bdds) [24] as the internal structure to represent the model. However in this section, for illustrative purposes only the model checking algorithm for Ctl initially developed by Queille and Sifakis [115] and Clarke, Emerson and Sistla [39] is presented.

Definition 5 (Model-checking problem) *Given a Kripke structure $K = \langle S, S_0, AP, L, TR \rangle$, representing a finite-state transition system, and a temporal logic formula f expressing a given specification, find the set of all states in S that satisfy f: $\{s \in S : M, s \vDash f\}$.*

2.3.1 Model checking Ctl

This algorithm relies on the fact that Ctl can only express state formulas, and thus the model checking problem can be solved by computing the sets of states satisfying the subformulas, without inspecting the execution paths in the Kripke structure. This is achieved by a recursive labeling algorithm which will work backwards from the states satisfying the atomic propositions contained in the formula to the initial state(s) and check if the formula is present on the labels of the initial state(s). Since some of the Ctl operators present some dualities and equivalences between themselves (Table 2.3), it is sufficient for the recursive labeling algorithm (Algorithm 1) to handle six cases, depending on whether the formula f is atomic or has one of the following forms: $\neg$, $\vee$, EX, EU and EG. Algorithm 2 and Algorithm 3 respectively handle the case of operators EU and EG.

Given a Ctl formula f and a Kripke structure K, the model checking Algorithm 1, by recursion on f, computes the set $label(s)$ for all the states $s \in S$. Upon termination, a state $s \vDash f$ if $f \in label(s)$. The formula f is valid on the Kripke structure K if and only if there is at least one state $s \in S_0$ such that $s \vDash f$.

The Ctl model-checking procedure is exemplified with a running example, by verifying the Ctl formula $f = \mathsf{E}[a\ \mathsf{U}\ b]$ on the state transition graph presented on Figure 2.1. The Figure 2.5 shows the computation tree presented in Figure 2.2 with a state numbering, before the application of the model checking algorithm.

Let us consider s_1 belonging to the set of initial states S_0. We first start by applying the Algorithm 1 to the formula f, which states that we have to recursively apply the same Algorithm 1 to each of the arguments of the Until operator, and then apply Algorithm 2 to

Algorithm 1 Procedure for labeling states satisfying f: $mc\_ctl(f)$

if $f = p$ **then**
  add p to $label(s)$ for those states s with $p \in L(s)$
else if $f = g_0 \vee g_1$ **then**
  $mc\_ctl(g_0); mc\_ctl(g_1)$; add f to all states labeled with g_0 or g_1
else if $f = \neg g$ **then**
  $mc\_ctl(g)$; add f to all states not labeled with g
else if $f = \mathsf{EX}\ g$ **then**
  $mc\_ctl(g)$; add f to all states with an TR-successor labeled by g
else if $f = \mathsf{E}[g_0\ \mathsf{U}\ g_1]$ **then**
  $mc\_ctl(g_0); mc\_ctl(g_1); check\_eu(g_0, g_1)$;
else if $f = \mathsf{EG}\ g$ **then**
  $mc\_ctl(g); check\_eg(g)$;
end if

Algorithm 2 Procedure for labeling states satisfying $\mathsf{E}[f\ \mathsf{U}\ g]$: $check\_eu(f, g)$

$T := \{s \mid g \in label(s)\}$;
for all $s \in T$ **do**
  $label(s) := label(s) \cup \{\mathsf{E}[f\ \mathsf{U}\ g]\}$;
end for
while $T \neq \emptyset$ **do**
  choose $s \in T$;
  $T := T \setminus \{s\}$;
  for all t **such that** $TR(t, s)$ **do**
    if $\mathsf{E}[f\ \mathsf{U}\ g] \notin label(t)$ **and** $f \in label(t)$ **then**
      $label(t) := label(t) \cup \{\mathsf{E}[f\ \mathsf{U}\ g]\}$;
      $T := T \cup \{t\}$;
    end if
  end for
end while

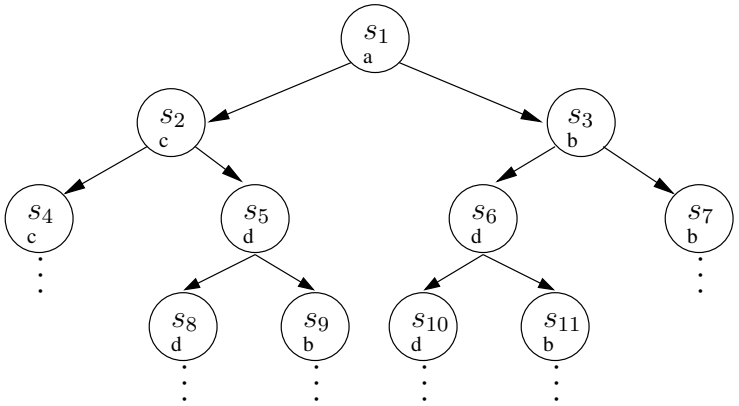


Figure 2.5: Computation tree for the state transition graph on Figure 2.1 with state numbers, before applying the labeling procedure.

Algorithm 3 Procedure for labeling states satisfying $\mathsf{EG}f$: $check\_eg(f)$

$S' := \{s \mid f \in label(s)\}$;
$SCC := \{C \mid C \text{ is a nontrivial SCC of } S'\}$;
$T := \bigcup_{C \in SCC} \{s \mid s \in C\}$;
for all $s \in T$ **do**
  $label(s) := label(s) \cup \{\mathsf{EG}\ f\}$;
end for
while $T \neq \emptyset$ **do**
  choose $s \in T$;
  $T := T \setminus \{s\}$;
  for all t **such that** $t \in S'$ **and** $TR(t, s)$ **do**
    if $\mathsf{EG}f \notin label(t)$ **then**
      $label(t) := label(t) \cup \{\mathsf{EG}\ f\}$;
      $T := T \setminus \{t\}$;
    end if
  end for
end while

the Until operator. Upon the first application, Algorithm 1 will label $s_i = \{a\}$ all the states s_i which have the atomic proposition a valid. Next, Algorithm 1 is applied to the second argument of the formula where all the states s_i with the atomic proposition b valid in that state will be labeled $s_i = \{b\}$. Finally, the Algorithm 2 is applied to the Until operator itself.

The application of Algorithm 2 will start with the definition of the set $T = \{s_3, s_7, s_9, s_{11}\}$, corresponding to the states that were previously labeled with the atomic proposition b. Next, the Until operator is added to the label of all the states in T. The state s_{11} is then picked and removed from the set T, and it is checked if any of its incoming states are labeled with the atomic proposition a and are not labeled with the Until operator. In this case, s_6 is an incoming state of s_{11} but the atomic proposition $a \notin label(s_6)$. Proceeding with the cycle, in the next two iterations pick the states s_7 and s_9 and remove them from the set T. They are both checked if any of their incoming states is labeled with the atomic proposition a and are not labeled with the Until operator. Also, in this case, only s_3 and s_5 are incoming states but without the atomic proposition a in their labels. The final state to be picked and removed from the set T is the state s_3. The state s_1 is an incoming state of the state s_3, labeled with the atomic proposition a and not yet labeled with the Until operator. The Until operator is therefore added the set $label(s_1)$ and this state is added to the set T. Continuing the cycle, the state s_1 is now picked and removed from the set T. Since there are no incoming states and no more states in the set T, the Algorithm 2 terminates and recursively falls back to the Algorithm 1. This algorithm also terminates, finalizing the state labeling procedure for the formula $f = \mathsf{E}[a\ \mathsf{U}\ b]$ on the state transition graph presented on Figure 2.1. The resulting labeled computation tree is presented in Figure 2.6.

In this example, it is verified if the transition system in Figure 2.1 satisfies formula $f =$

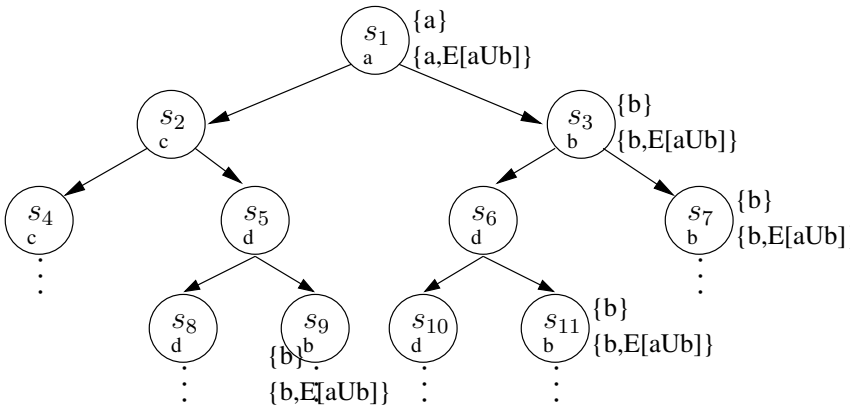


Figure 2.6: Computation tree for the state transition graph on Figure 2.1 with state numbers, after applying the labeling procedure for the formula $f = \mathsf{E}[a\ \mathsf{U}\ b]$.

$\mathsf{E}[a\ \mathsf{U}\ b]$. Like previously mentioned, the model checking procedure states that a given formula f is valid if and only if there is at least one state $s \in S_0$ such that $s \vDash f$. Since, in this example the formula $f = \mathsf{E}[a\ \mathsf{U}\ b]$ is present in the label of the initial state s_1, the model checking procedure returns true, meaning that the transition system satisfies the formula.

The description of the simple model-checking procedure for CTL is just a brief example of a simple automated algorithm for the verification of the validity of property in a system. The following section presents some examples of the application of this automated model-checking procedure for the verification of real biological systems.

2.4 Model checking in computational systems biology

Several applications of model checking in the field of systems biology have been proposed in recent years [4, 8, 9, 14, 15, 18, 21, 26, 29, 57, 64, 123]. These applications differ for example in the type of model that is being analyzed or the formalism used to represent the model. Examples of such formalisms are Boolean and other logical models [95, 124, 134], Petri nets [31, 64, 82], ordinary differential equations [20, 44], among others. In the sequel, we briefly review some of these applications.

Gilles Bernot *et al.* [18] applied model checking for the verification of a model of a genetic regulatory network illustrating mucus production [66] in *Pseudomonas aeruginosa*. These bacteria are commonly present in the environment and secrete mucus in lungs affected by cystic fibrosis, increasing the respiratory deficiency of the patient.

The authors were interested in the development of a multi-stable model, satisfying the experimental biological results, where one stable state regularly produces mucus and the other does not. The developed network is presented in Figure 2.7. The main regulator of mucus production is the gene *algU*. It codes for the protein AlgU which positively regulates both an

operon and all genes involved in the mucus synthesis. The operon is made up of *algU* itself (positive autoregulation) and 4 other genes which code for an inhibitor complex (anti-AlgU) of the protein AlgU.

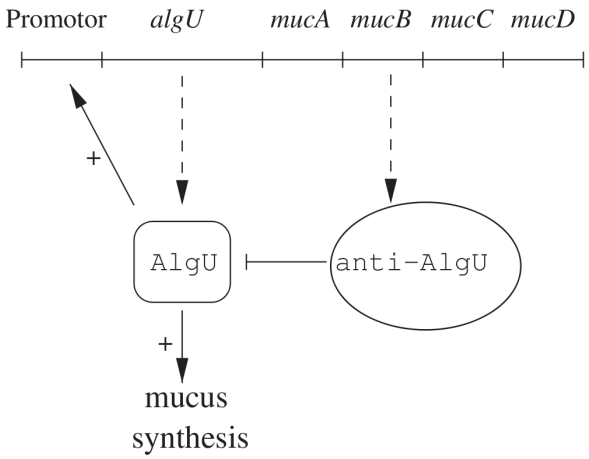


Figure 2.7: Main regulatory genes of mucus production in *Pseudomonas aeruginosa* [18].

To model this network the authors used the discrete multivalued approach developed by Thomas [132], obtaining an asynchronous state transition graph representing all the possible behaviors of the system, and the CTL temporal logic was chosen to describe these behaviors.

The existence of a stable state with a regular production of mucus, was expressed with the following CTL formula:

$$(x = 2) \Rightarrow \mathsf{AX}\ \mathsf{AF}\ (x = 2)$$

where x represents the concentration of mucus.

Moreover, the following CTL formula was used to test if the wild bacteria never produce mucus by themselves when starting from a basal state:

$$(x = 0) \Rightarrow \mathsf{AG}(\ \neg(x = 2))$$

The formal verification of these biological properties, was carried out using the SMV model checker [105]. The authors started from a set of 648 possible models that were cutted down to 4 models satisfying several criteria, including the two previous formulas. These 4 models correspond to 2 possible networks having the 2 desired stable states each.

Muffy Calder *et al.* [26] applied model checking for the analysis of a signal transduction pathway, where mitogenic and differentiation signals are transmitted from the cell membrane to the nucleus, in particular the RKIP inhibited ERK pathway [35] (Figure 2.8). The authors were interested in understanding the role of the kinase inhibitor protein RKIP in the behavior of the pathway.

To build the models, the authors considered discrete approximations of protein concentrations due to the lack of quantitative data. The models were based on high level descriptions of stochastic transition systems, in particular, continuous time Markov chains (CTMCs). Continuous Stochastic Logic (CSL) was used to express and check a variety of temporal queries for both transient and steady state behaviors.

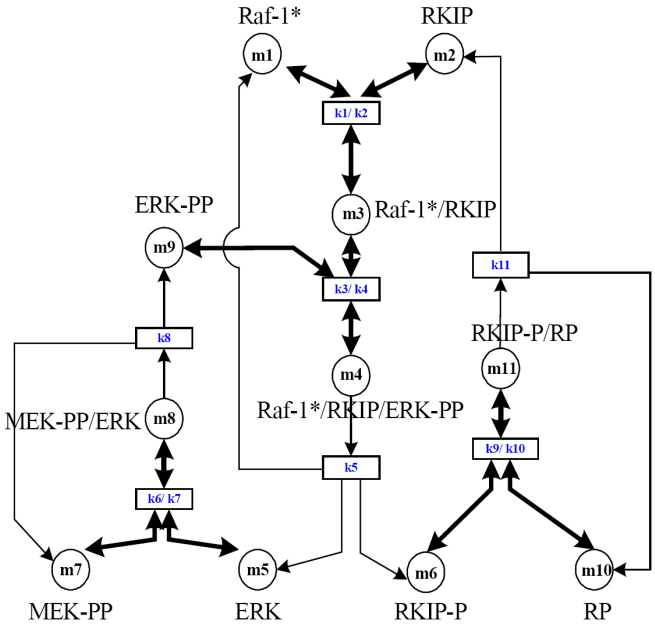


Figure 2.8: Graphical representation of the RKIP inhibited ERK pathway [35]. A circle represents a state for the concentration of a protein and a bar a kinetic parameter of reaction to be estimated. The directed arc (arrows) connecting a circle and a bar represents a direction of a signal flow. The bi-directional thick arrows represent a association and a dissociation rate at same time. The thin unidirectional arrows represent a production rate of products.

Several types of temporal properties were considered, like: steady state analysis of stability of a protein (*i.e.*, a protein reaches and then remains within certain bounds), steady state analysis of protein stability when varying reaction rates (*i.e.*, a protein is more likely to be stable for certain reaction rates) and transient analysis of protein activation sequence (*i.e.*, concentration peak ordering).

An example of such a property is: "What is the probability that the concentration of Raf-1*/RKIP/ERK-PP complex will be less than Level M until Raf-1*/RKIP complex reaches concentration Level C?", which corresponds to the following CSL formula[1]:

$$P =_? [(Raf-1^*/RKIP/ERK-PP < M) \mathsf{U} (Raf-1^*/RKIP = C)]$$

[1]Continuous Stochastic Logic (CSL) [6] was not introduced in this thesis, but the CSL formula above exemplifies the possibility of asking for the probability of a given formula being true rather than asking if it is true or false.

The models were described using the language of the PRISM model checker [88], where the reactions were synchronous processes and concentrations discrete, abstract quantities. The results were then compared with traditional ordinary differential equation-based simulations.

Chabrier and Fages [28] applied model checking for the analysis of the mammalian cell cycle control [83]. The authors were interested in characterizing the activity of two major types of proteins: cyclins and cyclin-dependent kinases. The cyclin-dependent kinases activity requires the binding to a cyclin, and it is controlled by specific inhibitors and by stimulatory or inhibitory phosphorylations by several kinases or phosphatases which in turn may produce positive feedback loops.

Similarly to [26], due to the lack of quantitative data the authors considered discrete approximations of protein concentrations, in this case, all variables were considered boolean. To model this network, the authors used concurrent transition systems, introduced by [122], for reasoning about concurrent programs. The chosen temporal logic was CTL, to capture the branching type behavior of the system.

An example of a biological property is "Can the cell reach a state s_1 while passing by another state s_2?", which corresponds to the following CTL formula:

$$\mathsf{EF}\ (s_2 \wedge \mathsf{EF}\ (s_1))$$

or "Is a certain (partially described) state s of the cell a stable state?", corresponding to the following CTL formula:

$$\mathsf{AG}\ s$$

To perform the verification of these biological queries, the authors used the symbolic model checking approach using the NUSMV model checker [37]. The authors were mainly interested in putting forward the advantages of using this kind of approach with respect to the use of just simulation techniques. In subsequent work, the authors applied these techniques to answer some biological questions (*e.g.*, Rizk *et al.* [116]).

2.5 Problems and limitations

Despite the existence of successful examples of the application of model-checking techniques to systems biology, several problems and limitations still persist, preventing these techniques from being naturally used by the systems biology community.

First, the expressiveness of the existing temporal logics is not sufficient to express all biological properties. Second, the formulation of biological properties as statements in temporal logic is not obvious, especially for non-expert users. And third, most of the existing modeling and simulation tools are not capable of applying model-checking techniques in a transparent way.

2.5.1 Expressiveness of standard temporal logics (CTL/LTL)

Definition 6 (Multistability) *Multistability is a property of a system containing multiple attractors, representing a choice on the evolution of the system from the set of initial states. Typically, one can force the system to alternate between these attractors by changing the input values of the system.*

Definition 7 (Oscillations) *In a Kripke structure $K = \langle S, S_0, AP, L, TR \rangle$, an oscillation is defined by a set of states $O \in S$ constituting a finite path $\pi_{i,j}$ $(i < j)$, where the set of atomic propositions characterizing state π_i is equal to the set of atomic propositions characterizing state π_j $(label(\pi_i) = label(\pi_j))$.*

Formal verification based on model-checking techniques has been mostly applied to the analysis of hardware and software systems. Most of the existing examples in these domains use classical temporal logics, like CTL [39] and LTL [99], to capture branching-time and linear-time behaviors, respectively. However, the application to actual biological systems brought to the fore the necessity of expressing both of these behaviors using a single temporal logic. In particular, the expression of multistability and other properties of non-deterministic models (corresponding to branching-time properties), as well as the expression of (damped) oscillations in the concentration of molecular species (corresponding to linear-time properties).

For example, the description of an oscillatory behavior between two genes g_1 and g_2 cannot be represented in CTL and is described using the following LTL formula:

$$\neg \mathsf{G}((inc\_g_1 \wedge \neg \mathsf{F}\ inc\_g_2) \vee (inc\_g_2 \wedge \neg \mathsf{F}\ dec\_g_1)$$
$$\vee\ (dec\_g_1 \wedge \neg \mathsf{F}\ dec\_g_2) \vee (dec\_g_2 \wedge \neg \mathsf{F}\ inc\_g_1))$$

where the atomic propositions inc_g_1, dec_g_1, inc_g_2 and dec_g_2 indicate the increasing and decreasing expression of genes g_1 and g_2, respectively.

An obvious solution would be to consider the previously described CTL* [51] language or the more expressive propositional μ-calculus [84], since both subsume CTL and LTL. However, these powerful temporal logics are complex to understand and use by non-experts.

The desirable situation would be to have a temporal logic at least as expressive as CTL* to allow the expression of properties of biological interest, while keeping a good compromise with user-friendliness.

2.5.2 Formulation of temporal logic properties by non-expert users

Despite the fact that formal verification based on model checking provides a powerful technology to query biological systems, finding good questions is not evident. The problem of posing relevant and interesting questions is critical in modeling in general, but even more so in the context of applying formal verification techniques, due to the fact that it is not easy

to formulate queries in temporal logic, especially for non-expert users who are not familiar with this kind of reasoning.

For instance, considering ψ_g to denote the expression of gene g, what does the following CTL formula specify?

$$\mathsf{EF}\ (\phi_g) \wedge \neg \mathsf{E}(\mathsf{true}\ \mathsf{U}\ (\neg([P] > 0.9\ \mu M) \wedge \mathsf{E}(\mathsf{true}\ \mathsf{U}\ \phi_g)))$$

This CTL formula specifies that "gene g is eventually expressed, and necessarily preceded over the whole duration of the experiment by a concentration larger than 0.9 μM of the transcription factor P".

A non-expert user will find it hard to formulate a biological question in temporal logic from scratch, since he is not familiar with the existing temporal logics, in particular their syntax and semantics. In this sense, what is missing is some sort of set of pre-structured biological questions that could serve as a basis for the inexperienced user.

In this sense, one step forward will be the definition of a set of pre-structured questions to be used to formulate the main biological questions initially performed by biologists during model analysis.

2.5.3 Computational infrastructure

Most of the existing modeling and simulation tools perform some kind of analysis of the model, like simulation, attractor search or some kind of reachability analysis, but are not capable of applying model-checking techniques in a transparent way or simply provide some kind of connection to model checkers. Some modeling tools like GINSIM [110] and previous versions of GNA [45] are capable of exporting the model in an implicit or explicit format accepted by the model checker, continuing the entire analysis in the model-checking environment, without any feedback from the modeling tool. An exception is the modeling tool BIOCHAM [27], which integrates the model checker NUSMV and allows for a more flexible iterative modeling and verification approach.

Current model checkers were developed by the formal verification community for the verification of hardware and software systems. Such model checkers have specific languages for describing a model and a property of a system, making their application difficult for the non-expert user from its usage. Additionally, a successful usage of a model checker still has to face the problem of interpretation of the verification result.

What is missing is the definition and development of a generic architecture capable of integrating modeling and simulation tools with formal verification tools, in a transparent way, in combination with a graphical user interface capable of helping the user with the visualization and interpretation of verification results.

Part II

Methods

Chapter 3

Computation Tree Regular Logic (CTRL)

The application to actual biological systems brought a few properties of the network dynamics to the fore that are not easily expressed in these logics. For instance, questions about multistability are important in the analysis of biological regulatory networks [49, 133], but difficult (or impossible) to express in LTL. CTL is capable of dealing with branching time, important for multistability and other properties of non-deterministic models. However, it does not do a good job when faced with questions about cycles in a Kripke structure. Such cycles may correspond to sustained or damped oscillations in the concentration of molecular species, underlying cellular rhythms [33, 94]. CTL is not expressive enough to specify the occurrence of oscillations of indefinite length, a special kind of fairness property [14]. An obvious solution would be to consider CTL* [51] or the propositional μ-calculus [84], both of which subsume CTL and LTL; however, these powerful branching-time logics are complex to understand and use by non-experts. More generally, temporal logics have difficulties in expressing experimental observations, which often take the form of patterns of events corresponding to variations of system parameters (protein concentrations, their derivatives, etc.). Observations are conveniently and concisely formulated in terms of regular expressions, but these are not provided by standard temporal logics such as CTL and LTL.

In this chapter, we describe a temporal specification language that allows expressing properties of biological interest and strikes a suitable compromise between expressive power, user-friendliness, and complexity of model checking. In order to achieve this objective, we propose a specification language named CTRL (*Computation Tree Regular Logic*), which extends CTL with regular expressions and fairness operators [102]. CTRL is more expressive than previous extensions of CTL with regular expressions, such as RCTL [17] and RegCTL [22], whilst having a simpler syntax due to a different choice of primitive temporal operators, inspired from dynamic logics like PDL (Propositional Dynamic Logic) [55]. CTRL also subsumes CTL, LTL, and PDL-Δ [126], allowing in particular the concise expression of bistability and oscillation

properties by using potentiality and fairness operators, respectively.

As regards the evaluation of CTRL formulas on Kripke structures, we attempt to avoid the effort of building a model checker from scratch by reusing as much as possible existing verification technology. We adopt as verification engine CADP [61], a state-of-the-art verification toolbox for concurrent asynchronous systems that provides, among other functionalities, on-the-fly model checking and diagnostic generation for μ-calculus formulas on labeled transition systems (LTSs). In order to reuse this technology, we have to move from the state-based setting (CTRL and Kripke structures) to the action-based setting (μ-calculus and LTSs). The translation from Kripke structures to LTSs is done in the standard way [40], simply by migrating information from states to transition labels without changing the structure of the model, *i.e.*, keeping the same states and transition relations. The translation from CTRL to an action-based logic is carried out by considering as target language HMLR [92], an alternative equational representation of the modal μ-calculus. Since HMLR is accepted as input by the EVALUATOR 3.6 [103] model checker of CADP, the development of a translator from CTRL to HMLR results in the immediate availability of an on-the-fly model checker equipped with full diagnostic features (generation of examples and counterexamples).

Section 3.1 defines the syntax and semantics of CTRL and discusses its expressiveness w.r.t. existing widely-used logics. Section 3.2 defines the regular equation systems (RESs), an intermediate equational form into which CTRL formulas will be translated. Subsections 3.2.2 and 3.2.3 present the translations from CTRL to RESs and then to modal equation systems (MESs). Section 3.3 describes the on-the-fly model checking procedure for CTRL, indicates its complexity, and shows its implementation in connection with CADP. Section 3.4 provides some concluding remarks.

3.1 CTRL definition

3.1.1 Syntax and semantics

In this section, it is assumed the existence of a Kripke structure $K = \langle S, S_0, AP, L, TR \rangle$ (see Section 2.1 for the formal definition), on which all formulas will be interpreted.

The syntax and semantics of CTRL are defined in Figure 3.1. The logic contains two kinds of entities: *state formulas* (noted φ) and *regular formulas* (noted ρ), which characterize properties of states and path intervals, respectively. State formulas are built from atomic propositions $p \in AP$ by using standard boolean operators and the EF, AF, EF^{∞}, AF^{∞} temporal operators indexed by regular formulas ρ. Regular formulas are built from state formulas by using standard regular expression operators.

The interpretation $[\![\varphi]\!]_K$ of a state formula denotes the set of states of the Kripke structure K that satisfy φ. The notation $\pi_{i,j}$ (where i $\leq$ j) stands for the interval going from the ith state of a path π to the jth state inclusively. The interpretation of regular formulas is

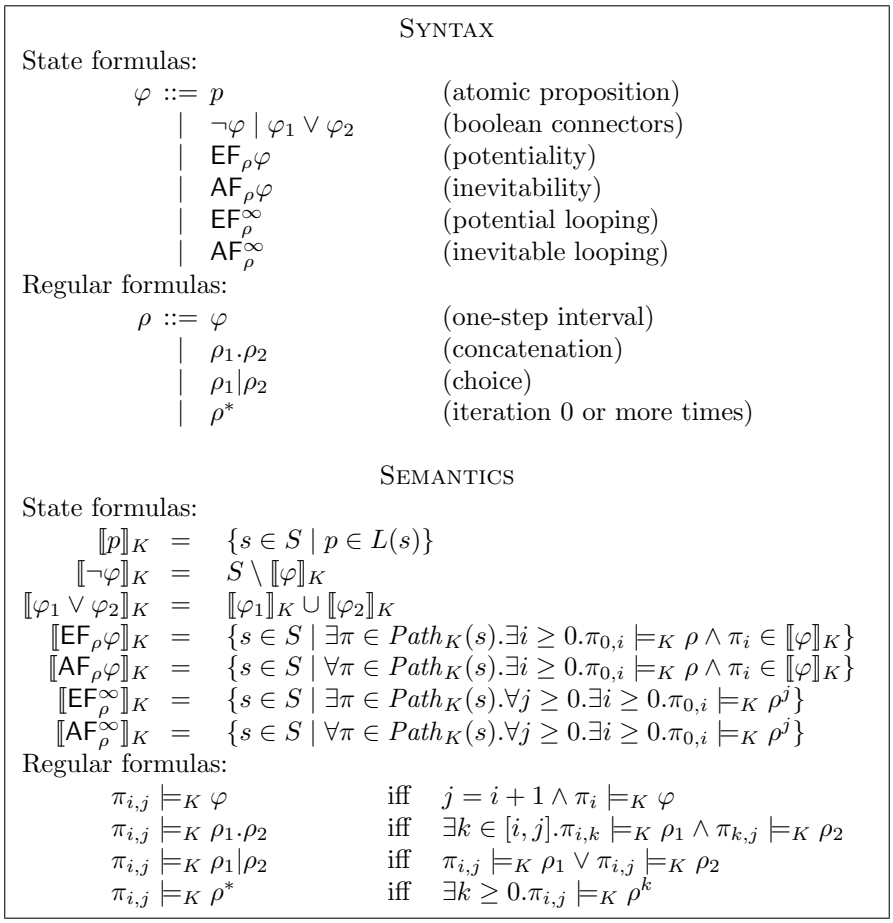


Figure 3.1: Syntax and semantics of CTRL.

defined by the satisfaction relation $\models_K$, which indicates whether an interval $\pi_{i,j}$ of a path in a Kripke structure K satisfies a regular formula ρ (notation $\pi_{i,j} \models_K \rho$). The notation ρ^j (where $j \geq 0$) stands for the concatenation $\rho \ldots \rho$, where ρ occurs j times. The semantics of boolean operators is defined in the standard way. A state satisfies the potentiality formula $\mathsf{EF}_\rho\varphi$ (resp. inevitability formula $\mathsf{AF}_\rho\varphi$) iff some (resp. all) of its outgoing paths contain a prefix satisfying ρ and lead to a state satisfying φ. A state satisfies the potential looping formula EF^∞_ρ (resp. the inevitable looping formula AF^∞_ρ) iff some (resp. all) of its outgoing paths consist of an infinite concatenation of intervals satisfying ρ. An interval satisfies the one-step interval formula φ iff it consists of two states, the first of which satisfies φ. An interval satisfies the concatenation formula $\rho_1.\rho_2$ if it is the concatenation of two subintervals, the first one satisfying ρ_1 and the second one satisfying ρ_2. An interval satisfies the choice formula $\rho_1|\rho_2$ iff it satisfies either ρ_1, or ρ_2. An interval satisfies the iteration formula ρ^* iff it is the concatenation of (0 or more) subintervals satisfying ρ. By definition, an empty interval $\pi_{i,i}$ satisfies ρ^0 for any regular formula ρ. A Kripke structure K satisfies a state formula φ

(notation $K \models \varphi$) iff there is at least one state $s \in S_0$ such that $s \in [\![\varphi]\!]_K$.

Figure 3.2 shows several derived operators on states and intervals defined in order to facilitate the specification of properties. The trajectory operator $\mathsf{EG}_\rho\varphi$ and the invariance operator $\mathsf{AG}_\rho\varphi$ are defined as duals of inevitability and potentiality operators, respectively, similarly to the their CTL counterparts (obtained by dropping the ρ formulas). They express that for some (resp. each) path going out of a state, all of its prefixes satisfying ρ lead to states satisfying φ. The potential saturation operator $\mathsf{EG}^\dashv_\rho$ and the inevitable saturation operator $\mathsf{AG}^\dashv_\rho$ express that some (resp. each) path going out of a state contains a prefix satisfying ρ^* such that no other larger prefix satisfies ρ^*; in other words, only a finite number of intervals satisfying ρ can be concatenated at the beginning of the path. The empty interval operator nil is defined as the iteration (0 or more times) of the false proposition; an interval satisfies the formula nil iff it contains a single state. The iteration (1 or more times) operator '+' is defined in the standard way; an interval satisfies ρ^+ iff it is the concatenation of (1 or more) intervals satisfying ρ.

$\mathsf{true} = p \vee \neg p$	(true, $p \in AP$)
$\mathsf{false} = \neg\mathsf{true}$	(false)
$\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$	(conjunction)
$\varphi_1 \Rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$	(implication)
$\varphi_1 \Leftrightarrow \varphi_2 = (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)$	(equivalence)
$\mathsf{EG}_\rho\varphi = \neg\mathsf{AF}_\rho\neg\varphi$	(trajectory)
$\mathsf{AG}_\rho\varphi = \neg\mathsf{EF}_\rho\neg\varphi$	(invariance)
$\mathsf{EG}^\dashv_\rho = \neg\mathsf{AF}^\infty_\rho$	(potential saturation)
$\mathsf{AG}^\dashv_\rho = \neg\mathsf{EF}^\infty_\rho$	(inevitable saturation)
$\mathsf{nil} = \mathsf{false}^*$	(empty interval)
$\rho^+ = \rho.\rho^*$	(iteration 1 or more times)

Figure 3.2: Derived (boolean, temporal, and regular) operators of CTRL.

To facilitate the manipulation of CTRL state formulas, we transform them in *positive normal form* (PNF) by propagating the negations downwards, using the rules in Figure 3.2, until they reach the atomic propositions p. For convenience, we also include in the set AP the negations of all propositions p, as well as the boolean constants true and false. State formulas in PNF are thus composed of atomic propositions, disjunctions and conjunctions, and all primitive and derived CTRL temporal operators defined in Figures 3.1 and 3.2.

3.1.2 Examples of temporal properties

We illustrate below the use of CTRL operators for specifying typical temporal properties of biological regulatory networks. The analogy with properties of communication protocols and concurrent systems is made explicit through the terminology of safety, liveness and fairness.

Safety properties Informally, these properties specify that "something bad never happens" during the functioning of the system. They can be expressed in CTRL by identifying the sequences of states corresponding to violations of the safe progression of execution, characterizing them using a regular formula ρ, and forbidding their existence in the Kripke structure by checking the formula $\mathsf{AG}_\rho\mathsf{false}$. For example, the CTRL formula below states that it is impossible to express cell cycle genes g_1 and g_2 in response to an external stress signal:

$$\mathsf{AG}_{\mathsf{true}^*.sig^+.(g_1|g_2)}\mathsf{false}$$

where the atomic proposition sig indicates the presence of the external signal, and g_1 and g_2 the expression of the cell cycle genes. This property can also be specified in CTL using two nested temporal operators:

$$\mathsf{AG}(sig \Rightarrow \neg\mathsf{E}[sig\ \mathsf{U}\ (g_1 \vee g_2)])$$

where $\mathsf{AG}\ \varphi = \neg\mathsf{E}[\mathsf{true}\ \mathsf{U}\ \neg\varphi]$ is the invariance operator of CTL.

Liveness properties Informally, these properties specify that "something good eventually happens" during the functioning of the system. They can be expressed in CTRL by capturing the desirable sequences of states, characterizing them using a regular formula ρ, and expressing their potential or inevitable presence in the Kripke structure using the EF_ρ and AF_ρ operators, respectively. For instance, the CTRL formula below states that every time a particular nutrient nut is present in the medium, it will eventually be taken up and consumed by the cell, as witnessed by the expression of gene g_{nut} coding for an appropriate transporter. This may be preceded by the expression of one or more genes in the set G, responsible for the uptake and consumption of other nutrients:

$$\mathsf{AG}_{\mathsf{true}^*.nut}\mathsf{AF}_{(\mathsf{true}^*.G)^*.g_{nut}}\mathsf{true}$$

where the atomic proposition G indicates that one or more genes in the corresponding set are expressed. This property cannot be specified in CTL because of the two nested $*$ operators in the regular subformula of AF.

Fairness properties Informally, these properties specify the progression of certain concurrent processes in the system, which are possibly antagonistic. In CTRL, fairness properties can be expressed by identifying the infinite sequences of events denoting the proper progression of a certain process, characterizing them using the EF^∞_ρ operator, and requiring their presence in the Kripke structure. The CTRL formula below captures the oscillatory expression patterns

of two genes g_1 and g_2 involved in the circadian rhythm:

$$\mathsf{EF}_{\mathsf{true}^*}\mathsf{EF}^{\infty}_{inc\_g_1.\mathsf{true}^*.inc\_g_2.\mathsf{true}^*.dec\_g_1.\mathsf{true}^*.dec\_g_2.\mathsf{true}^*}$$

where the atomic propositions inc_g_1, dec_g_1, inc_g_2, dec_g_2 indicate the increasing and decreasing expression of genes g_1 and g_2, respectively. This property is unexpressible in CTL because of the repeated alternation of inc_g_i and dec_g_i, but it can be stated in LTL using five temporal operators:

$$\neg\mathsf{G}((inc\_g_1 \wedge \neg\mathsf{F}inc\_g_2) \vee (inc\_g_2 \wedge \neg\mathsf{F}dec\_g_1)$$
$$\vee\ (dec\_g_1 \wedge \neg\mathsf{F}dec\_g_2) \vee (dec\_g_2 \wedge \neg\mathsf{F}inc\_g_1))$$

CTRL was designed such that fairness operators ($\mathsf{EF}^{\infty}_{\rho}$ and $\mathsf{AF}^{\infty}_{\rho}$) are at the same level as the other temporal operators of the logic. Compared to other logics, such as fair CTL [53], in which fairness constraints are added as side formulas modifying the interpretation of the temporal operators, we believe that an explicit presence of infinite looping operators in the logic allows a more direct and intuitive description of complex cycles (*e.g.*, matching regular expressions containing nested iteration operators) present in the behavior of genetic regulatory networks.

3.1.3 Expressiveness

CTRL is a natural extension of CTL [39], whose main temporal operators can be described using the EF and AF operators of CTRL as follows:

$$\mathsf{E}[\varphi_1\ \mathsf{U}\ \varphi_2] = \mathsf{EF}_{\varphi_1^*}\ \varphi_2 \qquad\qquad \mathsf{A}[\varphi_1\ \mathsf{U}\ \varphi_2] = \mathsf{AF}_{\varphi_1^*}\ \varphi_2$$

The until operator U of CTL is not primitive in CTRL; this is a difference w.r.t. other extensions of CTL, such as RCTL [17] and RegCTL [22], which keep the U operator primitive as in the original logic.

CTRL also subsumes LTL [99], because the potential looping operator EF^{∞} is able to capture the acceptance condition of Büchi automata. Assuming that the atomic proposition p characterizes the accepting states in a Büchi automaton (represented as a Kripke structure), the formula below expresses the existence of an infinite sequence passing infinitely often through an accepting state:

$$\mathsf{EF}^{\infty}_{\mathsf{true}^*.p.\mathsf{true}^+}$$

The + operator is necessary in order to avoid empty sequences consisting of a single state satisfying p. Of course, the EF^{∞} operator does not allow a direct encoding of LTL operators, but may serve as an intermediate form for LTL model checking; in this respect, EF^{∞} is similar to the "never claims" used for specifying properties in the early versions of the SPIN model

checker [73].

Thus, CTRL subsumes both CTL and LTL. This subsumption is strict, since these two logics are uncomparable w.r.t. their expressive power (*i.e.*, each one can describe properties unexpressible in the other one) [40]. In fact, the CTRL fragment containing the boolean connectors and the temporal operators EF and EF^{∞} can be seen as a state-based variant of PDL-Δ [126]. It was shown that this logic subsumes CTL*, whose operators can be encoded (although not in a succinct way) in PDL-Δ by means of the translation proposed first in [136] and reconsidered later in [43]. This subsumption is strict, because PDL-Δ can characterize sequences that match regular expressions containing nested iteration operators, such as $(a.b)^*.c$, which cannot be expressed in CTL* [52]. Since CTRL syntactically subsumes (a state-based variant of) PDL-Δ, it follows from the above expressiveness results that CTRL also subsumes CTL*.

As regards other existing extensions of CTL with regular operators, CTRL also subsumes RegCTL, whose U operator indexed by a regular formula can be expressed using the EF operator of CTRL as follows:

$$\mathsf{E}[\varphi_1 \; \mathsf{U}^{\rho} \; \varphi_2] = \mathsf{EF}_{\rho \; \& \; \varphi_1^*}\varphi_2$$

The & operator stands for the intersection of regular formulas; although this operator is not present in CTRL, its occurrence above can be expanded in terms of the regular operators available in CTRL by applying the rules below:

$$\begin{array}{ll} \varphi' \; \& \; \varphi^* = \varphi' \; \& \; \varphi & (\rho_1.\rho_2) \; \& \; \varphi^* = (\rho_1 \; \& \; \varphi^*).(\rho_2 \; \& \; \varphi^*) \\ (\rho_1|\rho_2) \; \& \; \varphi^* = (\rho_1 \; \& \; \varphi^*)|(\rho_2 \; \& \; \varphi^*) & ({\rho_1}^*) \; \& \; \varphi^* = (\rho_1 \; \& \; \varphi^*)^* \end{array}$$

The subsumption of RegCTL is strict because the U operator of RegCTL cannot describe an infinite concatenation of intervals satisfying a regular formula ρ, which is specified in CTRL using the $\mathsf{EF}^{\infty}_{\rho}$ operator. Brázdil and Cerná [22] show that RegCTL is more expressive than RCTL [17], the extension of CTL with regular expressions underlying the SUGAR [16] specification language; consequently, CTRL also subsumes RCTL.

3.2 Translation from CTRL to modal equation systems

Building an efficient model checker for a branching-time temporal logic equipped with regular expressions, such as CTRL, is a complex and time-consuming task. Here we aim at facilitating this task by reusing as much as possible existing verification technology available in the field of concurrent systems, namely the CADP toolbox [61]. A model checker for CTRL can thus be obtained by translating this logic into HMLR [92], one of the input languages accepted by CADP.

This technical section is devoted to the translation of CTRL state formulas into modal equation systems (MESS), which are the state-based counterpart of HMLR. Using such a

translation to obtain a model checking procedure for a temporal logic with regular constructs is not common practice, most of the existing procedures for this kind of logics being based on automata [17, 22]. Therefore, we describe this translation in sufficient detail and we illustrate it with examples. Readers from the bioinformatics and systems biology field that are not necessarily interested in the technical details of temporal logic translations can safely skip Section 3.2 and go directly to Section 3.3 to get an overall view of the CTRL model checker.

The translation of a CTRL state formula φ into a MES involves two steps: first the formula is translated into a regular equation system (RES), and then the RES is transformed into a MES. These two steps are purely syntactic, *i.e.*, they do not depend upon the Kripke structure on which the formulas and the equation systems are interpreted. We first define the syntax and semantics of RESs and MESs, and then we detail the two translation steps. Appendix A contains the proofs of the translation phases from CTRL to MESs.

3.2.1 Regular and modal equation systems

To apply our model checking method, we need to translate CTRL state formulas into an equational representation, which is more suitable than the tree-like representation underlying the syntax definition in Figure 3.1. To achieve this, we first need to extend the grammar of CTRL state formulas with propositional variables $X \in \mathcal{X}$, which denote sets of states:

$$\varphi ::= X \mid p \mid \ldots$$

Propositional variables are interpreted w.r.t. a Kripke structure K by an environment $\delta : \mathcal{X} \rightarrow 2^S$, which is a partial function mapping propositional variables to state sets. The interpretation of state formulas must be extended to handle propositional variables: $[\![\varphi]\!]_K\delta$ denotes the set of states satisfying φ in the context of δ, which must map every variable occurring in φ to a state set. The interpretation of propositional variables is defined as follows: $[\![X]\!]_K\delta = \delta(X)$. The interpretation of the other state formulas defined in Figure 3.1 remains unchanged, except that an extra parameter δ is added to the interpretation $[\![\ ]\!]$. The translation to MESs ensures that all occurrences of propositional variables in state formulas are positive, *i.e.*, they fall under an even number of negations. This *syntactic monotonicity* condition was proposed initially to ensure the well-definedness of propositional μ-calculus formulas [84].

As intermediate language for translating CTRL state formulas, we use *regular equation systems* (RESs), which are the propositional counterpart of the PDLR (PDL with recursion) specifications introduced Mateescu and Sighireanu [103]. The syntax and semantics of RESs are defined in Figure 3.3. Equation blocks B are sets of fixed point equations having propositional variables $X \in \mathcal{X}$ in the left-hand sides and CTRL state formulas (possibly containing propositional variables) in the right-hand sides. All equations of a block have the same fixed

point sign $\sigma \in \{\mu, \nu\}$, where μ and ν denote minimal and maximal fixed points, respectively. The free (fv) and bound (bv) variables in an equation block list are defined as follows:

$$
\begin{array}{ll}
fv(\varepsilon) = \emptyset & bv(\varepsilon) = \emptyset \\
fv(B.BL) = (fv(B) \setminus bv(BL)) \cup fv(BL) & bv(B.BL) = bv(B) \cup bv(BL) \\
fv(\{X_i \stackrel{\sigma}{=} \varphi_i\}_{1 \leq i \leq n}) = \bigcup_{i=1}^{n} fv(\varphi_i) & bv(\{X_i \stackrel{\sigma}{=} \varphi_i\}_{1 \leq i \leq n}) = \{X_1, ..., X_n\}
\end{array}
$$

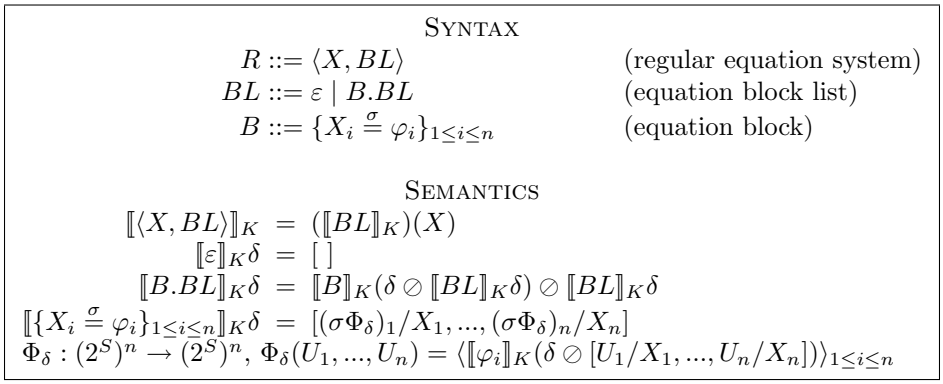


Figure 3.3: Syntax and semantics of regular equation systems.

The set $fv(\varphi_i)$ contains all propositional variables occurring in φ_i. A block list BL is *closed* if $fv(BL) = \emptyset$. We consider that all nonempty block lists $B.BL$ satisfy the following conditions: $bv(B) \cap bv(BL) = \emptyset$ (*normal form*) and $fv(B) \subseteq bv(B) \cup bv(BL)$ (*alternation-free*). In a block list $B.BL$, block B depends upon another block B' of BL if $fv(B) \cap bv(B') \neq \emptyset$, *i.e.*, B contains a free variable bound in B'. The alternation-free condition means that there are no cyclic dependencies between equation blocks, and block B depends only upon the blocks of BL, placed at his right in the list $B.BL$. In a Res $R = \langle X, BL \rangle$, BL is assumed to be nonempty and closed. X is called the *main variable* and must be bound in the first block of BL.

The interpretation of a Res $R = \langle X, BL \rangle$ on a Kripke structure $K = \langle S, S_0, AP, L, TR \rangle$ is the value of variable X as obtained by solving the block list BL. The interpretation $[\![BL]\!]_K \delta$ of a block list in the context of an environment δ is another environment assigning state sets to all variables bound in BL. Since the blocks of BL depend upon each other from left to right, the interpretation of BL can be defined inductively, by solving the blocks from right to left. The notation $\delta \oslash [U_1/X_1, ..., U_n/X_n]$ stands for the extension of δ with $[U_1/X_1, ..., U_n/X_n]$, *i.e.*, an environment identical to δ except for variables $X_1, ..., X_n$, which are mapped to the state sets $U_1, ..., U_n$, respectively. The empty environment is noted $[\ ]$. The interpretation of an equation block B is the environment mapping the variables bound in B to the state sets given by the corresponding fixed point of the functional associated to the block. When BL is closed, the δ environment is omitted. The state formulas in the right-hand sides of equations

are assumed to be syntactically monotonic, which according to Tarski's theorem [128] ensures the well-definedness of the functionals associated to blocks.

A *modal equation system* (MES) $M = \langle X, BL \rangle$ is a RES where all CTRL temporal operators occurring in the right-hand sides of equations contain only atomic regular formulas, *i.e.*, without any regular operator ('.', '|', '*'). MESs are the propositional counterpart of the HMLR specifications, proposed by Larsen [92] as an equational form of the modal μ-calculus.

3.2.2 Translation to regular equation systems

The translation of a CTRL state formula φ into a RES is defined by the syntactic function $t(\varphi) = \langle t_X(\varphi), t_{BL}(\varphi) \rangle$ given in Figure 3.4. The two components $t_X(\varphi)$ and $t_{BL}(\varphi)$ denote the main variable and the equation block list produced by $t(\varphi)$, respectively. For each translation rule, X denotes a "fresh" propositional variable, different from all the other variables contained in φ and in $t(\varphi)$. The notation $BL_1; BL_2$ indicates the concatenation of two equation block lists BL_1, BL_2 and is defined inductively as follows: $\varepsilon; BL_2 = BL_2$, and $(B.BL_1); BL_2 = B.(BL_1; BL_2)$.

For simplicity, in the translation of propositional constants we omitted the empty block list, *i.e.*, we wrote $\{X \stackrel{\mu}{=} p\}$ instead of $\{X \stackrel{\mu}{=} p\}.\varepsilon$. If φ is closed, then the block list produced by the translation is also closed, *i.e.*, $bv(t_{BL}(\varphi)) = \emptyset$. The translation given in Figure 3.4 preserves the interpretation of formulas, as stated by the proposition below.

Proposition 1 (Translation from CTRL to RESs) *Let K be a Kripke structure and φ a state formula of* CTRL. *Then:*

$$[\![\varphi]\!]_K \delta = [\![t(\varphi)]\!]_K \delta$$

for any propositional environment δ.

$$
\begin{aligned}
t(p) &= \langle X, \{X \stackrel{\mu}{=} p\} \rangle \\
t(\varphi_1 \vee \varphi_2) &= \langle X, \{X \stackrel{\mu}{=} t_X(\varphi_1) \vee t_X(\varphi_2)\}.(t_{BL}(\varphi_1); t_{BL}(\varphi_2)) \rangle \\
t(\varphi_1 \wedge \varphi_2) &= \langle X, \{X \stackrel{\mu}{=} t_X(\varphi_1) \wedge t_X(\varphi_2)\}.(t_{BL}(\varphi_1); t_{BL}(\varphi_2)) \rangle \\
t(\mathsf{EF}_\rho \varphi) &= \langle X, \{X \stackrel{\mu}{=} \mathsf{EF}_\rho t_X(\varphi)\}.t_{BL}(\varphi) \rangle \\
t(\mathsf{AF}_\rho \varphi) &= \langle X, \{X \stackrel{\mu}{=} \mathsf{AF}_\rho t_X(\varphi)\}.t_{BL}(\varphi) \rangle \\
t(\mathsf{EG}_\rho \varphi) &= \langle X, \{X \stackrel{\nu}{=} \mathsf{EG}_\rho t_X(\varphi)\}.t_{BL}(\varphi) \rangle \\
t(\mathsf{AG}_\rho \varphi) &= \langle X, \{X \stackrel{\nu}{=} \mathsf{AG}_\rho t_X(\varphi)\}.t_{BL}(\varphi) \rangle \\
t(\mathsf{EF}^\infty_\rho) &= \langle X, \{X \stackrel{\nu}{=} \mathsf{EF}_\rho X\} \rangle \\
t(\mathsf{AF}^\infty_\rho) &= \langle X, \{X \stackrel{\nu}{=} \mathsf{AF}_\rho X\} \rangle \\
t(\mathsf{EG}^\dashv_\rho) &= \langle X, \{X \stackrel{\mu}{=} \mathsf{EG}_\rho X\} \rangle \\
t(\mathsf{AG}^\dashv_\rho) &= \langle X, \{X \stackrel{\mu}{=} \mathsf{AG}_\rho X\} \rangle
\end{aligned}
$$

Figure 3.4: Translation of CTRL state formulas into RESs.

To illustrate the translation of CTRL formulas into RESs, we consider the *bistability property* [49, 134], which specifies that after an initial state, two different equilibrium states can be potentially reached. This branching-time property can be expressed in CTRL as follows:

$$\mathsf{AG}_{\mathsf{true}^*.init}(\mathsf{EF}_{\mathsf{true}^*}\, eql_1 \wedge \mathsf{EF}_{\mathsf{true}^*}\, eql_2)$$

where the atomic propositions $init$, eql_1 and eql_2 denote the initial state and the two equilibrium states, respectively. By applying the translation defined in Figure 3.4 to this formula, we obtain the RES below:

$$\begin{array}{l}\langle X, \{X \stackrel{\nu}{=} \mathsf{AG}_{\mathsf{true}^*.init} Y\}.\{Y \stackrel{\mu}{=} Z_1 \wedge Z_2\}. \\ \quad \{Z_1 \stackrel{\mu}{=} \mathsf{EF}_{\mathsf{true}^*} U_1\}.\{U_1 \stackrel{\mu}{=} eql_1\}.\{Z_2 \stackrel{\mu}{=} \mathsf{EF}_{\mathsf{true}^*} U_2\}.\{U_2 \stackrel{\mu}{=} eql_2\}.\varepsilon\rangle\end{array}$$

The ';' operator produced by translating $\mathsf{EF}_{\mathsf{true}^*}\, eql_1 \wedge \mathsf{EF}_{\mathsf{true}^*}\, eql_2$ was expanded in terms of the '.' operator using the definition of ';'.

The size (number of variables and operators) of the RES $t(\varphi)$ produced by the translation is linear in the size (number of operators) of the formula φ, because every rule in Figure 3.4 creates, for each operator present in φ, one block containing a single equation with one operator in its right-hand side.

For simplicity, the translation of a state formula φ given in Figure 3.4 does not take care of the state subformulas ψ that may occur inside the regular formulas ρ. However, these subformulas must also be translated into RESs in order to be evaluated on a Kripke structure K by the model checking procedure. This is done by applying the translation recursively on every subformula ψ of a regular formula ρ, yielding an additional RES $t(\psi) = \langle t_X(\psi), t_{BL}(\psi)\rangle$. In practice, the block list $t_{BL}(\psi)$ of each additional RES $t(\psi)$ is concatenated to the block list $t_{BL}(\varphi)$ of the RES $t(\varphi)$, and the main variable $t_X(\psi)$ replaces the occurrence of the corresponding subformula ψ, as illustrated by the formula $\mathsf{EF}_{(\mathsf{AG}_{\mathsf{true}^*} p)^*} q$, whose translation yields the RES $\langle X, \{X \stackrel{\mu}{=} \mathsf{EF}_{Y^*} Z\}.\{Z \stackrel{\mu}{=} q\}.\{Y \stackrel{\nu}{=} \mathsf{AG}_{\mathsf{true}^*} U\}.\{U \stackrel{\nu}{=} p\}.\varepsilon\rangle$.

However, in order to simplify notations, we can exploit the fact that the RESs produced by translating the subformulas ψ are closed, and hence their main variables can be evaluated independently from the RES $t(\varphi)$. This allows to safely replace each subformula ψ by a "fresh" atomic proposition p_ψ, whose interpretation on K is obtained by evaluating the main variable $t_X(\psi)$ of the RES $t(\psi)$. On the example above, the RES becomes $\langle X, \{X \stackrel{\mu}{=} \mathsf{EF}_{r^*} Z\}.\{Z \stackrel{\mu}{=} q\}.\varepsilon\rangle$, where r has the same interpretation as the variable Y of the additional RES $\langle Y, \{Y \stackrel{\nu}{=} \mathsf{AG}_{\mathsf{true}^*} U\}.\{U \stackrel{\nu}{=} p\}.\varepsilon\rangle$. Therefore, in the sequel we will restrict ourselves to RESs in which the regular formulas occurring in the right-hand sides of equations are built only upon atomic propositions.

3.2.3 Translation to modal equation systems

Let $B = \{X_i \stackrel{\sigma}{=} \varphi_i\}_{1 \leq i \leq n}$ be an equation block. An equation block $\{X_n \stackrel{\sigma}{=} \psi_n, Y_j \stackrel{\sigma}{=} \psi_j\}_{n<j\leq m}$ is *suitable* for the substitution of equation $X_n \stackrel{\sigma}{=} \varphi_n$ if $fv(\psi_n) \cup \bigcup_{j=n+1}^{m} fv(\psi_j) = fv(\varphi_n)$ and $\bigcup_{i=1}^{n} fv(\varphi_i) \cap \{Y_{n+1}, ..., Y_m\} = \emptyset$. The notation $\{X_i \stackrel{\sigma}{=} \varphi_i\}_{1\leq i\leq n}[X_n \stackrel{\sigma}{=} \varphi_n := X_n \stackrel{\sigma}{=} \psi_n, Y_j \stackrel{\sigma}{=} \psi_j\}_{n<j\leq m}]$ represents the syntactic substitution of the equation $X_n \stackrel{\sigma}{=} \varphi_n$ by the equations $\{X_n \stackrel{\sigma}{=} \psi_n, Y_j \stackrel{\sigma}{=} \psi_j\}_{n<j\leq m}$ in B. This definition of substitution, which allows to replace only the last equation of a block, is general enough: since all equations of a block have the same fixed point sign, their order does not influence the values of the variables defined in the block, and therefore any equation of the block can be substituted by bringing it in the last position.

The translation of a RES equation block into a corresponding MES equation block is performed by repeatedly applying various transformations, most of them being substitutions of equations.

3.2.3.1 Operators EF_ρ and AG_ρ

In order to translate the equation blocks of the form $\{X \stackrel{\mu}{=} \mathsf{EF}_\rho Y\}$ and $\{X \stackrel{\nu}{=} \mathsf{AG}_\rho Y\}$ into MESs, we eliminate the regular expressions ρ by repeatedly applying appropriate substitutions. Each equation containing an EF_ρ or AG_ρ operator in its right-hand side is substituted with a suitable equation block containing simpler regular formulas, as defined in Figure 3.5 (Z and U are "fresh" propositional variables). The application of any substitution given in Figure 3.5 preserves the interpretation of equation blocks, as stated by the proposition below.

EQUATION	SUBSTITUTION BLOCK
$X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1.\rho_2} Y$	$\{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} Z, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho_2} Y\}$
$X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1\|\rho_2} Y$	$\{X \stackrel{\mu}{=} Z \vee U, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} Y, U \stackrel{\mu}{=} \mathsf{EF}_{\rho_2} Y\}$
$X \stackrel{\mu}{=} \mathsf{EF}_{\rho^*} Y$	$\{X \stackrel{\mu}{=} Y \vee Z, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho} X\}$
$X \stackrel{\nu}{=} \mathsf{AG}_{\rho_1.\rho_2} Y$	$\{X \stackrel{\nu}{=} \mathsf{AG}_{\rho_1} Z, Z \stackrel{\nu}{=} \mathsf{AG}_{\rho_2} Y\}$
$X \stackrel{\nu}{=} \mathsf{AG}_{\rho_1\|\rho_2} Y$	$\{X \stackrel{\nu}{=} Z \wedge U, Z \stackrel{\nu}{=} \mathsf{AG}_{\rho_1} Y, U \stackrel{\nu}{=} \mathsf{AG}_{\rho_2} Y\}$
$X \stackrel{\nu}{=} \mathsf{AG}_{\rho^*} Y$	$\{X \stackrel{\nu}{=} Y \wedge Z, Z \stackrel{\nu}{=} \mathsf{AG}_{\rho} X\}$

Figure 3.5: Substitutions for the EF_ρ and AG_ρ operators.

Proposition 2 (Substitution of EF and AG) *Let K be a Kripke structure and $B_1 = \{X_i \stackrel{\mu}{=} \varphi_i\}_{1\leq i\leq n}$, $B_2 = \{X_i \stackrel{\nu}{=} \varphi_i\}_{1\leq i\leq n}$ be two equation blocks. Then, for any propositional environment δ, the interpretation of B_1 (resp. B_2) w.r.t. δ does not change when a substitution given in the upper part (resp. the lower part) of Figure 3.5 is applied.*

By repeatedly applying these substitutions, all occurrences of regular operators in the right-hand sides of the equations can be eliminated. For the RES of the bistability property,

this translation yields the following MES:

$$\begin{array}{l}\langle X, \{X \stackrel{\nu}{=} Y_1 \wedge Y_2,\ Y_1 \stackrel{\nu}{=} \mathsf{AG}_{init} Y,\ Y_2 \stackrel{\nu}{=} \mathsf{AG}_{\mathsf{true}} X\}.\{Y \stackrel{\mu}{=} Z_1 \wedge Z_2\}. \\ \quad \{Z_1 \stackrel{\mu}{=} U_1 \vee Z_3,\ Z_3 \stackrel{\mu}{=} \mathsf{EF}_{\mathsf{true}} Z_1\}.\{U_1 \stackrel{\mu}{=} eql_1\}. \\ \quad \{Z_2 \stackrel{\mu}{=} U_2 \vee Z_4,\ Z_4 \stackrel{\mu}{=} \mathsf{EF}_{\mathsf{true}} Z_2\}.\{U_2 \stackrel{\mu}{=} eql_2\}.\varepsilon\rangle\end{array}$$

The equation block $\{X \stackrel{\nu}{=} \mathsf{AG}_{\mathsf{true}^*.init} Y\}$ was translated by successively applying the first and the third substitutions in the lower part of Figure 3.5.

The size of the MES equation block resulting from the translation of a RES equation block B of the form $\{X \stackrel{\mu}{=} \mathsf{EF}_\rho Y\}$ (resp. $\{X \stackrel{\nu}{=} \mathsf{AG}_\rho Y\}$) remains linear w.r.t. the size of B (and hence linear w.r.t. the size of the initial CTRL formula φ), since each substitution in Figure 3.5 replaces a regular operator by at most two variables and two temporal operators EF (resp. AG).

3.2.3.2 Operators AF_ρ and EG_ρ

The translation of the equation blocks $\{X \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$ and $\{X \stackrel{\nu}{=} \mathsf{EG}_\rho Y\}$ into MESs is more complicated than the translation of their EF_ρ and AG_ρ counterparts, because the substitutions given in Figure 3.5 to eliminate the regular expressions ρ are no longer valid for the AF_ρ and EG_ρ operators. We consider below only blocks of the form $\{X \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$, the processing of their EG_ρ counterparts being dual.

The translation of the $\{X \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$ equation blocks into MESs consists of three steps. First, the RES is temporarily transformed in *potentiality* form $\{X \stackrel{\mu}{=} \mathsf{EF}_\rho Y\}$ and subsequently translated into a potentiality MES by eliminating the regular expression ρ using the substitutions given in Section 3.2.3.1. Then, the resulting MES is transformed in *guarded* form, by eliminating all unguarded (*i.e.*, not preceded by a temporal operator) occurrences of variables in the right-hand sides of equations. Finally, the guarded MES is *determinized*, by replacing all occurrences of EF operators in the right-hand sides of equations by appropriate occurrences of AF operators in order to retrieve the interpretation of the initial equation block $\{X \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$. We will illustrate each step of the translation on the following equation block:

$$\{X \stackrel{\mu}{=} \mathsf{AF}_{(q|p^*)^*.(qr^*).(p^*|q^*)} Y\}.$$

Translation to potentiality form The difficulty of translating an equation block $\{X \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$ into a MES stems from the fact that *all* transition sequences going out of a state have to satisfy ρ before reaching a state satisfying Y, whereas the substitutions in Figure 3.5 allow to eliminate ρ on individual sequences only. To avoid this difficulty, we switch temporarily to the potentiality form $\{X \stackrel{\mu}{=} \mathsf{EF}_\rho Y\}$, we eliminate ρ by applying the substitutions, and we continue working with the resulting potentiality MES, which characterizes the existence of individual sequences satisfying ρ. The size of this MES is linear w.r.t. the size of the initial

block $\{X \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$, as stated in Section 3.2.3.1. Figure 3.6 shows the potentiality MES obtained from the equation block above by switching to potentiality form and applying the substitutions in Figure 3.5 (all equations have the sign μ, omitted for simplicity).

$X = \mathsf{EF}_{(q\|p^*)^*} Z_1$	$X = Z_3 \vee Z_1$			$X = Z_3 \vee Z_1$
	$Z_3 = \mathsf{EF}_{q\|p^*} X$	$Z_3 = Z_4 \vee Z_5$		$Z_3 = Z_4 \vee Z_5$
		$Z_5 = \mathsf{EF}_q X$		$Z_5 = \mathsf{EF}_q X$
		$Z_4 = \mathsf{EF}_{p^*} X$	$Z_4 = Z_6 \vee X$	$Z_4 = Z_6 \vee X$
			$Z_6 = \mathsf{EF}_p Z_4$	$Z_6 = \mathsf{EF}_p Z_4$
$Z_1 = \mathsf{EF}_{qr^*} Z_2$	$Z_1 = \mathsf{EF}_q Z_7$			$Z_1 = \mathsf{EF}_q Z_7$
	$Z_7 = \mathsf{EF}_{r^*} Z_2$	$Z_7 = Z_8 \vee Z_2$		$Z_7 = Z_8 \vee Z_2$
		$Z_8 = \mathsf{EF}_r Z_7$		$Z_8 = \mathsf{EF}_r Z_7$
$Z_2 = \mathsf{EF}_{p^*\|q^*} Y$	$Z_2 = Z_9 \vee Z_{10}$			$Z_2 = Z_9 \vee Z_{10}$
	$Z_9 = \mathsf{EF}_{p^*} Y$	$Z_9 = Z_{11} \vee Y$		$Z_9 = Z_{11} \vee Y$
		$Z_{11} = \mathsf{EF}_p Z_9$		$Z_{11} = \mathsf{EF}_p Z_9$
	$Z_{10} = \mathsf{EF}_{q^*} Y$	$Z_{10} = Z_{12} \vee Y$		$Z_{10} = Z_{12} \vee Y$
		$Z_{12} = \mathsf{EF}_q Z_{10}$		$Z_{12} = \mathsf{EF}_q Z_{10}$

Figure 3.6: Translation of $\{X \stackrel{\mu}{=} \mathsf{AF}_{(q|p^*)^*.(qr^*).(p^*|q^*)} Y\}$ to a potentiality MES.

The right-hand sides of the equations of the potentiality MES may contain unguarded occurrences of propositional variables (*i.e.*, not preceded by any EF operator), such as variable Z_1 in the equation $X = Z_3 \vee Z_1$. These occurrences will be eliminated in the next step of the translation.

Translation to guarded form The translation of a potentiality MES to guarded form eliminates all unguarded occurrences of variables in the right-hand sides of equations using the lemma below.

Lemma 1 (Absorption) *Let K be a Kripke structure and $B = \{X_i \stackrel{\mu}{=} \varphi_i\}_{1 \leq i \leq n}$ be an equation block such that $\varphi_n = X_n \vee \varphi$ and $X_n \notin fv(\varphi)$. Then:*

$$[\![\{X_i \stackrel{\mu}{=} \varphi_i\}_{1 \leq i \leq n}[X_n \stackrel{\mu}{=} X_n \vee \varphi := X_n \stackrel{\mu}{=} \varphi]]\!]_K \delta = [\![\{X_i \stackrel{\mu}{=} \varphi_i\}_{1 \leq i \leq n}]\!]_K \delta$$

for any propositional environment δ.

The equations of a potentiality MES, produced by the rules in Figure 3.5, have two possible forms: either unguarded (*i.e.*, containing disjunctions of variables in their right-hand side), or guarded (*i.e.*, containing a single occurrence of an EF operator in their right-hand side). The elimination of unguarded occurrences of variables is carried out by Algorithm 4. The first loop of the algorithm applies the absorption lemma and the idempotency of disjunction on each unguarded equation defining a variable X in order to eliminate the unguarded occurrences of X, and afterwards expands inline all unguarded occurrences of X in all the other equations of the MES. After executing the first loop on the potentiality MES given in Figure 3.6, we

obtain the MES shown in Figure 3.7. Upon termination of the first loop, the formulas in the right-hand sides of equations may contain only unguarded occurrences of Y and of variables X defined by guarded equations of the MES. The second loop of the algorithm expands inline those variables, thus eliminating all unguarded occurrences except those of Y. The result of applying the second loop on the MES in Figure 3.7 yields the MES shown in Figure 3.8.

Algorithm 4 Translation of a potentiality MES to guarded form

for all unguarded equations $X \stackrel{\mu}{=} \bigvee_j X_j$ **do**
  Eliminate X among X_j by applying the absorption lemma
  for all unguarded occurrences of X in the rsh of other equations **do**
    Substitute X by $\bigvee_j X_j$
  end for
end for
for all guarded equations $X \stackrel{\mu}{=} \mathsf{EF}_p X_j$ **do**
  Substitute X by $\mathsf{EF}_p X_j$ in all unguarded equations
end for

Initial list of unguarded eqns.	1^{st} loop of Algorithm 4	
	Var.	Updated equations
$X = Z_3 \vee Z_1$	X:	$Z_4 = Z_6 \vee Z_3 \vee Z_1$
$Z_2 = Z_9 \vee Z_{10}$	Z_2:	$Z_7 = Z_8 \vee Z_9 \vee Z_{10}$
$Z_3 = Z_4 \vee Z_5$	Z_3:	$X = Z_4 \vee Z_5 \vee Z_1$
$Z_4 = Z_6 \vee X$		$Z_4 = Z_6 \vee \cancel{Z_4} \vee Z_5 \vee Z_1$
$Z_7 = Z_8 \vee Z_2$	Z_4:	$X = Z_6 \vee Z_5 \vee Z_1 \vee \cancel{Z_5} \vee \cancel{Z_1}$
$Z_9 = Z_{11} \vee Y$		$Z_3 = Z_1 \vee Z_5 \vee Z_6 \vee \cancel{Z_5}$
$Z_{10} = Z_{12} \vee Y$	Z_9:	$Z_2 = Z_{11} \vee Y \vee Z_{10}$
		$Z_7 = Z_8 \vee Z_{11} \vee Y \vee Z_{10}$
	Z_{10}:	$Z_2 = Z_{11} \vee Y \vee Z_{12} \vee \cancel{Y}$
		$Z_7 = Z_8 \vee Z_{11} \vee Y \vee Z_{12} \vee \cancel{Y}$

Figure 3.7: Translation of a potentiality MES to guarded form (1^{st} part).

The guarded MESs obtained by applying Algorithm 4 can be further simplified by eliminating duplicate and unreachable equations. In the MES shown in Figure 3.8, the equations defining X, Z_3 and Z_4 have identical right-hand sides, and therefore variables Z_4 and Z_3 can be replaced by X and their equations deleted. Also, some of the variables will no longer be referenced after these substitutions, and therefore their equations can be safely removed. Finally, variables can be renamed in order to have a proper numbering, leading to the MES shown in Figure 3.9. This guarded MES is equivalent to $\{X \stackrel{\mu}{=} \mathsf{EF}_{(q|p^*)^*.(qr^*).(p^*|q^*)} Y\}$, the potentiality form of our running example. Intuitively, each variable defined by this MES denotes the suffix of a transition sequence in the Kripke structure satisfying the regular formula indexing the EF operator. In this respect, guarded potentiality MESs are similar to the equation systems defining the derivatives of regular expressions [25].

Equations after the 1^{st} loop	2^{nd} loop of Algorithm 4	
	Var.	Updated equations
$X = Z_6 \vee Z_5 \vee Z_1$	Z_1:	$X = Z_6 \vee Z_5 \vee \mathsf{EF}_q Z_7$
$Z_2 = Z_{11} \vee Y \vee Z_{12}$		$Z_3 = \mathsf{EF}_q Z_7 \vee Z_5 \vee Z_6$
$Z_3 = Z_1 \vee Z_5 \vee Z_6$		$Z_4 = Z_6 \vee Z_5 \vee \mathsf{EF}_q Z_7$
$Z_4 = Z_6 \vee Z_5 \vee Z_1$	Z_5:	$X = Z_6 \vee \mathsf{EF}_q X \vee \mathsf{EF}_q Z_7$
$Z_7 = Z_8 \vee Z_{11} \vee Y \vee Z_{12}$		$Z_3 = \mathsf{EF}_q Z_7 \vee \mathsf{EF}_q X \vee Z_6$
$Z_9 = Z_{11} \vee Y$		$Z_4 = Z_6 \vee \mathsf{EF}_q X \vee \mathsf{EF}_q Z_7$
$Z_{10} = Z_{12} \vee Y$	Z_6:	$X = \mathsf{EF}_p Z_4 \vee \mathsf{EF}_q X \vee \mathsf{EF}_q Z_7$
$Z_1 = \mathsf{EF}_q Z_7$		$Z_3 = \mathsf{EF}_q Z_7 \vee \mathsf{EF}_q X \vee \mathsf{EF}_p Z_4$
$Z_5 = \mathsf{EF}_q X$		$Z_4 = \mathsf{EF}_p Z_4 \vee \mathsf{EF}_q X \vee \mathsf{EF}_q Z_7$
$Z_6 = \mathsf{EF}_p Z_4$	Z_8:	$Z_7 = \mathsf{EF}_r Z_7 \vee Z_{11} \vee Y \vee Z_{12}$
$Z_8 = \mathsf{EF}_r Z_7$	Z_{11}:	$Z_2 = \mathsf{EF}_p Z_9 \vee Y \vee Z_{12}$
$Z_{11} = \mathsf{EF}_p Z_9$		$Z_7 = \mathsf{EF}_r Z_7 \vee \mathsf{EF}_p Z_9 \vee Y \vee Z_{12}$
$Z_{12} = \mathsf{EF}_q Z_{10}$		$Z_9 = \mathsf{EF}_p Z_9 \vee Y$
	Z_{12}:	$Z_2 = \mathsf{EF}_p Z_9 \vee Y \vee \mathsf{EF}_q Z_{10}$
		$Z_7 = \mathsf{EF}_r Z_7 \vee \mathsf{EF}_p Z_9 \vee Y \vee \mathsf{EF}_q Z_{10}$
		$Z_{10} = \mathsf{EF}_q Z_{10} \vee Y$

Figure 3.8: Translation of a potentiality MES to guarded form (2^{nd} part).

$$\left\{\begin{array}{ll} X & = \mathsf{EF}_p X \vee \mathsf{EF}_q X \vee \mathsf{EF}_q Z_7 \\ Z_7 & = \mathsf{EF}_r Z_7 \vee \mathsf{EF}_p Z_9 \vee \mathsf{EF}_q Z_{10} \vee Y \\ Z_{10} & = \mathsf{EF}_q Z_{10} \vee Y \\ Z_9 & = \mathsf{EF}_p Z_9 \vee Y \end{array}\right\} \quad \left\{\begin{array}{l} X_1 = \mathsf{EF}_p X_1 \vee \mathsf{EF}_q X_1 \vee \mathsf{EF}_q X_2 \\ X_2 = \mathsf{EF}_p X_4 \vee \mathsf{EF}_q X_3 \vee \mathsf{EF}_r X_2 \vee Y \\ X_3 = \mathsf{EF}_q X_3 \vee Y \\ X_4 = \mathsf{EF}_p X_4 \vee Y \end{array}\right\}$$

Figure 3.9: Guarded potentiality MES after simplifications (left) and renaming (right).

The guarded potentiality MESs produced by applying Algorithm 4 have at most the same number of variables as the original MESs, but may present in the worst-case a quadratic increase in the number of operators. However, we observed in practice that the number of variables in the guarded MESs is much smaller than in the original MESs (thanks to elimination of redundant equations) and the number of operators remains close to linear w.r.t. the original MESs, and hence w.r.t. the size of the initial CTRL formula.

Determinization The last step of the translation consists in determinizing the guarded potentiality MES obtained so far in order to obtain a MES with the same meaning as the initial RES $\{X \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$. Consider the following potentiality MES in guarded form:

$$\left\{ X_i \stackrel{\mu}{=} \bigvee_{j=1}^{n} (h_{ij} \wedge \mathsf{EF}_{p_{ij}} X_j) \vee (h_i \wedge Y) \right\}_{1 \leq i \leq n}$$

where $h_{ij}, h_i \in \mathbf{Bool}$ and $p_{ij} \in AP$ for all $1 \leq i, j \leq n$. The coefficients h_{ij} and h_i allow to simplify notations: only the terms $\mathsf{EF}_{p_{ij}} X_j$ with their coefficients h_{ij} equal to true (and

similarly for the unguarded occurrences of Y with their h_i equal to true) are present in the right-hand sides of equations. An equation defining variable X_i is said to have the index i. Note that the translation to guarded potentiality form may produce equations containing guarded occurrences of Y, *e.g.*, formulas $\mathsf{EF}_p Y$ in their right-hand sides; in this case, bringing the MES to the form above requires to introduce an extra equation $X_{n+1} \stackrel{\mu}{=} Y$ and to replace by X_{n+1} all guarded occurrences of Y (but not its unguarded occurrences). The *determinized* MES corresponding to the guarded potentiality MES above is defined as follows:

$$\left\{ X_I \stackrel{\mu}{=} \bigvee_{\emptyset \subset Q \subseteq prop(I)} \mathsf{AF}_Q X_{vars(Q,I)} \vee (h(I) \wedge Y) \right\}_{I \subseteq [1,n]}$$

where:

- $prop(I) \stackrel{d}{=} \{p_{ij} \mid i \in I \wedge j \in [1,n] \wedge h_{ij}\}$ is the set of atomic propositions occurring as subscripts of EF operators in the equations of the guarded potentiality MES having their index in the set I.
- $vars(Q,I) \stackrel{d}{=} \{j \in [1,n] \mid \exists i \in I.(h_{ij} \wedge p_{ij} \in Q)\}$ is the set of indexes of propositional variables which occur in the right-hand side of some equation having its index in the set I and whose corresponding EF operator is subscripted by some atomic proposition contained in the set Q.
- $h(I) \stackrel{d}{=} \exists i \in I.h_i$ is equal to true iff Y occurs unguarded in some equation having its index in the set I.

In the AF operators of the determinized MES, the subscript Q stands for the conjunction of all the atomic propositions contained in the set Q.

The determinization restores the meaning of the initial equation block $\{X \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$, as stated by the proposition below.

Proposition 3 (Determinization correctness) *Let K be a Kripke structure, $R = \{X_1 \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$ an equation block, and M the* MES *obtained from R after translation in guarded potentiality form and determinization. Then:*

$$([\![M]\!]_K \delta)(X_{\{1\}}) = ([\![R]\!]_K \delta)(X_1)$$

for any propositional environment δ.

Figure 3.10 shows the determinized version of the guarded potentiality MES produced by the previous translation phases from the equation block $\{X \stackrel{\mu}{=} \mathsf{AF}_{(q|p^*)^*.(qr^*).(p^*|q^*)} Y\}$. For conciseness, we represent index sets just by concatenating their elements, *e.g.*, the set $\{1,2,3\}$ is denoted by 123. We observe that this MES can be simplified by eliminating

duplicate equations (*e.g.*, the equations defining variables X_{12}, X_{123}, X_{124} and those defining $X_2, X_{23}, X_{24}, X_{234}$) and by absorbing certain operands using the identity $\mathsf{AF}_p X_I \vee \mathsf{AF}_{pq} X_I = \mathsf{AF}_p X_I$, yielding the MES on the left of Figure 3.11. Finally, the right-hand side formulas of some equations may occur as subformulas in other equations and can therefore be replaced by their corresponding left-hand side variables, leading to the final determinized MES shown on the right of Figure 3.11. In practice, these simplifications can be carried out incrementally as the equations are generated, avoiding the complete construction of the determinized MES prior to simplification. Moreover, sometimes it is possible to determine statically whether certain atomic propositions are mutually exclusive, which allows to remove the AF operators whose index subformulas contain those propositions together.

$$\left\{\begin{array}{ll}
X_1 & \stackrel{\mu}{=} \mathsf{AF}_p X_1 \vee \mathsf{AF}_q X_{12} \vee \mathsf{AF}_{pq} X_{12} \\
X_{12} & \stackrel{\mu}{=} \mathsf{AF}_p X_{14} \vee \mathsf{AF}_q X_{123} \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pq} X_{1234} \vee \mathsf{AF}_{pr} X_{124} \vee \mathsf{AF}_{qr} X_{123} \vee \mathsf{AF}_{pqr} X_{1234} \vee Y \\
X_{123} & \stackrel{\mu}{=} \mathsf{AF}_p X_{14} \vee \mathsf{AF}_q X_{123} \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pq} X_{1234} \vee \mathsf{AF}_{pr} X_{124} \vee \mathsf{AF}_{qr} X_{123} \vee \mathsf{AF}_{pqr} X_{1234} \vee Y \\
X_{124} & \stackrel{\mu}{=} \mathsf{AF}_p X_{14} \vee \mathsf{AF}_q X_{123} \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pq} X_{1234} \vee \mathsf{AF}_{pr} X_{124} \vee \mathsf{AF}_{qr} X_{123} \vee \mathsf{AF}_{pqr} X_{1234} \vee Y \\
X_{14} & \stackrel{\mu}{=} \mathsf{AF}_p X_{14} \vee \mathsf{AF}_q X_{12} \vee \mathsf{AF}_{pq} X_{124} \vee Y \\
X_{1234} & \stackrel{\mu}{=} \mathsf{AF}_p X_{14} \vee \mathsf{AF}_q X_{123} \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pq} X_{1234} \vee \mathsf{AF}_{pr} X_{124} \vee \mathsf{AF}_{qr} X_{123} \vee \mathsf{AF}_{pqr} X_{1234} \vee Y \\
X_2 & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee \mathsf{AF}_q X_3 \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pq} X_{34} \vee \mathsf{AF}_{pr} X_{24} \vee \mathsf{AF}_{qr} X_{23} \vee \mathsf{AF}_{pqr} X_{234} \vee Y \\
X_{23} & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee \mathsf{AF}_q X_3 \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pq} X_{34} \vee \mathsf{AF}_{pr} X_{24} \vee \mathsf{AF}_{qr} X_{23} \vee \mathsf{AF}_{pqr} X_{234} \vee Y \\
X_{234} & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee \mathsf{AF}_q X_3 \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pq} X_{34} \vee \mathsf{AF}_{pr} X_{24} \vee \mathsf{AF}_{qr} X_{23} \vee \mathsf{AF}_{pqr} X_{234} \vee Y \\
X_{24} & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee \mathsf{AF}_q X_3 \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pq} X_{34} \vee \mathsf{AF}_{pr} X_{24} \vee \mathsf{AF}_{qr} X_{23} \vee \mathsf{AF}_{pqr} X_{234} \vee Y \\
X_3 & \stackrel{\mu}{=} \mathsf{AF}_q X_3 \vee Y \\
X_{34} & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee \mathsf{AF}_q X_3 \vee \mathsf{AF}_{pq} X_{34} \vee Y \\
X_4 & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee Y
\end{array}\right\}$$

Figure 3.10: Determinized MES produced from $\{X \stackrel{\mu}{=} \mathsf{AF}_{(q|p^*)^*.(qr^*).(p^*|q^*)} Y\}$.

$$\left\{\begin{array}{ll}
X_1 & \stackrel{\mu}{=} \mathsf{AF}_p X_1 \vee \mathsf{AF}_q X_{12} \\
X_{12} & \stackrel{\mu}{=} \mathsf{AF}_p X_{14} \vee \mathsf{AF}_q X_{12} \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pr} X_{12} \vee Y \\
X_{14} & \stackrel{\mu}{=} \mathsf{AF}_p X_{14} \vee \mathsf{AF}_q X_{12} \vee Y \\
X_2 & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee \mathsf{AF}_q X_3 \vee \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pq} X_{34} \vee Y \\
X_3 & \stackrel{\mu}{=} \mathsf{AF}_q X_3 \vee Y \\
X_{34} & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee \mathsf{AF}_q X_3 \vee \mathsf{AF}_{pq} X_{34} \vee Y \\
X_4 & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee Y
\end{array}\right\} \qquad \left\{\begin{array}{ll}
X_1 & \stackrel{\mu}{=} \mathsf{AF}_p X_1 \vee \mathsf{AF}_q X_{12} \\
X_{12} & \stackrel{\mu}{=} \mathsf{AF}_r X_2 \vee \mathsf{AF}_{pr} X_{12} \vee X_{14} \\
X_{14} & \stackrel{\mu}{=} \mathsf{AF}_p X_{14} \vee \mathsf{AF}_q X_{12} \vee Y \\
X_2 & \stackrel{\mu}{=} \mathsf{AF}_r X_2 \vee X_{34} \\
X_{34} & \stackrel{\mu}{=} \mathsf{AF}_{pq} X_{34} \vee X_3 \vee X_4 \\
X_3 & \stackrel{\mu}{=} \mathsf{AF}_q X_3 \vee Y \\
X_4 & \stackrel{\mu}{=} \mathsf{AF}_p X_4 \vee Y
\end{array}\right\}$$

Figure 3.11: Determinized MES of $\{X \stackrel{\mu}{=} \mathsf{AF}_{(q|p^*)^*.(qr^*).(p^*|q^*)} Y\}$ after simplifications.

The determinization of a guarded potentiality MES defined above is similar to the subset construction procedure used for determinizing finite automata [1]. In the worst-case, the size of the determinized MES resulting from the translation of an equation block $\{X \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$

is exponential w.r.t. the size (number of operators and atomic propositions) of the regular formula ρ. However in practice, the size of determinized MESs obtained after simplifications is close to linear w.r.t. the size of ρ, as illustrated by the final MES shown in Figure 3.11. When ρ is *deterministic* (*i.e.*, each atomic proposition occurs only once in the right-hand side of each equation of the guarded potentiality MES used as intermediate form, and all atomic propositions are mutually exclusive on the states of the Kripke structure), the size of the resulting determinized MES remains linear w.r.t. the size of ρ.

3.2.3.3 Operators $\mathsf{EF}^{\infty}_{\rho}$, $\mathsf{AF}^{\infty}_{\rho}$, $\mathsf{EG}^{\dashv}_{\rho}$, and $\mathsf{AG}^{\dashv}_{\rho}$

According to the rules given in Fig 3.4, the $\mathsf{EF}^{\infty}_{\rho}$ and $\mathsf{AF}^{\infty}_{\rho}$ operators are translated into equation blocks of the form $\{X \stackrel{\nu}{=} \mathsf{EF}_{\rho} X\}$ and $\{X \stackrel{\nu}{=} \mathsf{AF}_{\rho} X\}$, respectively. The interpretation of these equation blocks is given by $\nu\Phi_e$ and $\nu\Phi_a$, where the functionals $\Phi_e, \Phi_a : 2^S \rightarrow 2^S$ are defined as follows:

$$\Phi_e(U) = [\![\mathsf{EF}_{\rho} X]\!]_K[U/X] = ([\![\{X_1 \stackrel{\mu}{=} \mathsf{EF}_{\rho} X\}]\!]_K[U/X])(X_1)$$
$$\Phi_a(U) = [\![\mathsf{AF}_{\rho} X]\!]_K[U/X] = ([\![\{X_1 \stackrel{\mu}{=} \mathsf{AF}_{\rho} X\}]\!]_K[U/X])(X_1).$$

The evaluation of the $\mathsf{EF}^{\infty}_{\rho}$ and $\mathsf{AF}^{\infty}_{\rho}$ operators requires to compute the maximal fixed points of the functionals Φ_e and Φ_a, which are defined as the minimal fixed points of the functionals associated to the RESs $R_e = \{X_1 \stackrel{\mu}{=} \mathsf{EF}_{\rho} X\}$ and $R_a = \{X_1 \stackrel{\mu}{=} \mathsf{AF}_{\rho} X\}$. Therefore, these operators belong to $L\mu_2$, the μ-calculus fragment of alternation depth 2 [52], which allows one level of mutual recursion between minimal and maximal fixed points. The operators $\mathsf{EG}^{\dashv}_{\rho}$ and $\mathsf{AG}^{\dashv}_{\rho}$ are handled dually w.r.t. $\mathsf{AF}^{\infty}_{\rho}$ and $\mathsf{EF}^{\infty}_{\rho}$, respectively.

3.3 An on-the-fly model checker for CTRL

The translation from CTRL to MESs presented in Section 3.2 provides the basis of a model checking procedure, which was implemented by reusing as much as possible the on-the-fly verification technology available in the CADP toolbox [61]. The resulting CTRL model checker was coupled with the qualitative simulation tool GNA [14], which was enhanced in order to allow the specification of biological properties as temporal logic formulas. We outline below the principles of the implementation of the CTRL model checker.

3.3.1 Implementation

The most direct way of obtaining a model checker for CTRL was to take advantage of existing verification technology. As verification engine, we use CADP[1] (*Construction and Analysis of Distributed Processes*) [61], a state-of-the-art verification toolbox for concurrent

[1] `http://www.inrialpes.fr/vasy/cadp`

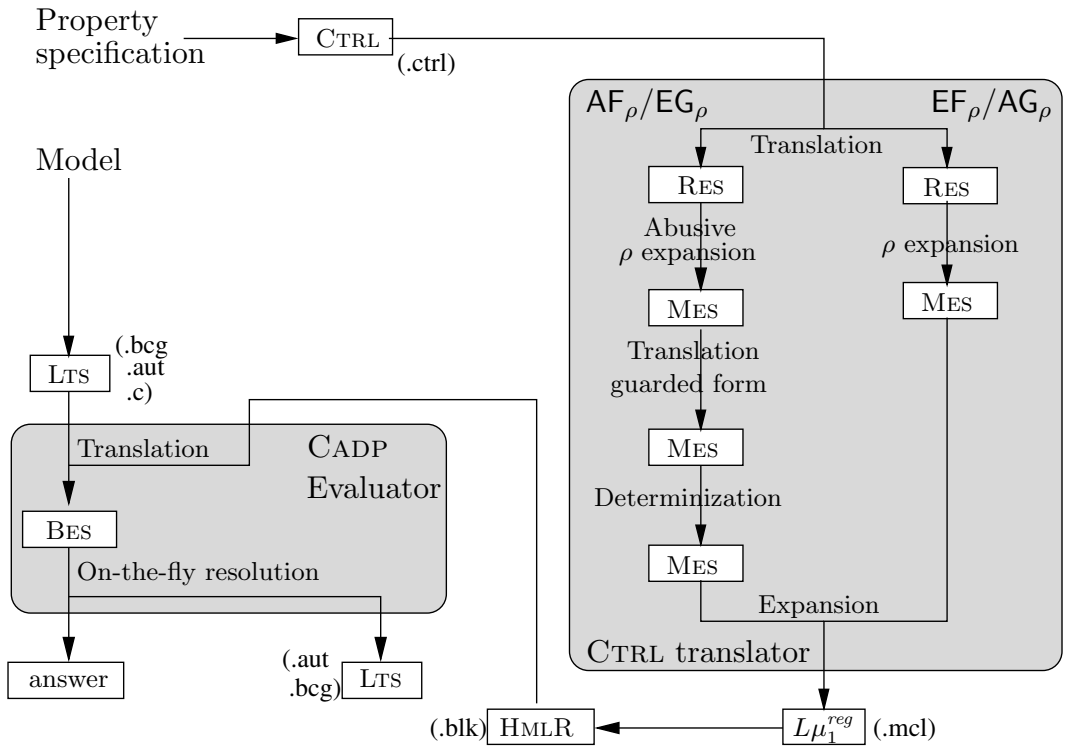


Figure 3.12: Model checker for CTRL. CTRL formulas are translated into HMLR specifications, which can be evaluated by the on-the-fly model checker EVALUATOR 3.6.

asynchronous systems. CADP offers a wide range of functionalities assisting the user throughout the design process: compilation and rapid prototyping, random execution, interactive and guided simulation, model checking and equivalence checking, test generation, and performance evaluation. The toolbox accepts as input process algebraic descriptions in LOTOS [76] or CHP [100], as well as networks of communicating automata in the EXP language [91].

The tools of CADP operate on labeled transition systems (LTSs), which are represented either explicitly (by their list of transitions) as compact binary files encoded in the BCG (*Binary Coded Graphs*) format, or implicitly (by their successor function) as C programs compliant with the OPEN/CÆSAR interface [60]. CADP contains the on-the-fly model checker EVALUATOR 3.6 [103], which evaluates regular alternation-free μ-calculus ($L\mu_1^{reg}$) formulas on implicit LTSs. The tool works by translating the verification problem in terms of the local resolution of a boolean equation system, which is performed using the algorithms available in the generic CÆSAR_SOLVE library [101]. EVALUATOR 3.6 uses HMLR as intermediate language: $L\mu_1^{reg}$ formulas are translated into HMLR specifications, whose evaluation on implicit LTSs can be straightforwardly encoded as a local boolean equation system resolution [41, 103]. The tool also generates full diagnostics (examples and counterexamples) illustrating the truth value of the formulas.

In order to reuse Evaluator 3.6, we had the choice of translating Ctrl formulas either to $L\mu_1^{reg}$ formulas, or to HmlR specifications. We adopted the second solution because it leads to a more succinct translation and avoids the translation step from $L\mu_1^{reg}$ to HmlR present in Evaluator 3.6. This technical choice motivated the definition of the translation from Ctrl to Mess in the first place. The architecture of the Ctrl translator (about 12,000 lines of code) is shown in Figure 3.12. The tool takes as input a Ctrl state formula and translates it to a Mes following the phases described in Section 3.2, which are different for the EF_ρ and AF_ρ operators and their dual counterparts. The Mes obtained is then converted into a HmlR specification by expanding the basic Ctrl temporal operators in terms of Hml modalities as described in [102].

3.3.2 Complexity

Table 3.1 summarizes the complexity of our model checking procedure for Ctrl. The EF_ρ and EF^∞_ρ operators, together with their respective duals AG_ρ and $\mathsf{AG}^\dashv_\rho$, are evaluated in linear-time w.r.t. the size of the formula and the size of the Kripke structure by applying the boolean equation system resolution algorithms given in [101, 104]. Moreover, the evaluation of these operators has a memory complexity $O(|\rho| \cdot |S|)$, meaning that only the states (and not the transitions) of the Kripke structure are stored. This fragment of Ctrl is the state-based counterpart of Pdl-Δ [126], which is more expressive than Ctl* [51]. Of course, this does not yield a linear-time model checking procedure for Ctl* (nor for its fragment Ltl), because the translation from Ctl* to Pdl-Δ is not succinct [136]. However, the linear-time evaluation of the EF^∞_ρ operator allows an efficient detection of complex cycles describing oscillation properties [29].

Operator		Complexity	
		ρ deterministic	ρ nondeterministic
EF_ρ	AG_ρ	$O(\|\rho\| \cdot (\|S\| + \|T\|))$	
AF_ρ	EG_ρ	$O(\|\rho\| \cdot (\|S\| + \|T\|))$	$O(2^{\|\rho\|} \cdot (\|S\| + \|T\|))$
EF^∞_ρ	$\mathsf{AG}^\dashv_\rho$	$O(\|\rho\| \cdot (\|S\| + \|T\|))$	
AF^∞_ρ	$\mathsf{EG}^\dashv_\rho$	$O(\|\rho\| \cdot (\|S\| + \|T\|))$	$O(2^{2\|\rho\|} \cdot (\|S\| + \|T\|)^2)$

Table 3.1: Complexity of model checking Ctrl operators on $K = \langle S, S_0, AP, L, TR \rangle$.

The AF_ρ operator and its dual EG_ρ are evaluated in linear-time only when the regular subformula ρ is deterministic (*i.e.*, without any choice operator '|' in the regular subformula ρ). In the general case, these operators are evaluated in exponential-time w.r.t. the size of ρ (because of the determinization phase) but still in linear-time in the size of the Kripke structure. In practice, the size of temporal formulas is much smaller than the size of Kripke structures, which reduces the impact of the factor $2^{|\rho|}$ on the total cost of model checking. Finally, the AF^∞_ρ operator and its dual $\mathsf{EG}^\dashv_\rho$ are evaluated in linear-time when ρ is deterministic

by applying a symmetric version of the boolean equation system resolution algorithm in [104]; in the general case, these operators are evaluated in doubly exponential-time w.r.t. the size of ρ and in quadratic-time w.r.t. the size of the Kripke structure. This complexity seems difficult to lower, since the boolean equation systems produced by translating these operators are of alternation depth 2 and have a general shape (arbitrary amounts of disjunctive and conjunctive equations).

3.4 Conclusions

Applications of model checking in systems biology have demonstrated its usefulness for understanding the dynamic behavior of regulatory networks in living cells, but also outlined certain limitations in expressiveness and user-friendliness. This work aims at alleviating these limitations in order to promote the practical usage of model checking in the bioinformatics and systems biology communities. The temporal logic CTRL that we proposed, an extension of CTL with regular expressions and fairness operators, allows a natural and concise description of typical properties of biological interest, such as the presence of multistability or oscillations in the concentrations of molecular species.

The extension of classical temporal logics with regular language constructs to increase their expressiveness and user-friendliness is a long-standing line of research. One of the first proposals in this direction was ETL [137], an extension of LTL with regular grammars, which is strictly more expressive than LTL while still having the same complexity of evaluation on Kripke structures. Another manner of increasing expressiveness is to enhance temporal operators with automata on infinite sequences; this was attempted for CTL [68] and CTL* [135]. Despite their expressive power, these extensions are difficult to implement and use in practice because of their complex syntax.

A more user-friendly approach, which led to successful implementations, is to index temporal operators by regular expressions instead of automata. FORSPEC [5] and EAGLE [10] are extensions of LTL with regular expressions and data handling mechanisms, dedicated respectively to hardware and runtime verification. RCTL [17] is an extension of CTL with regular expressions, which served subsequently as basis for the SUGAR [16] and PSL [75] specification languages used for hardware verification. RegCTL [22] is another extension of CTL with regular expressions, more expressive than RCTL, obtained by indexing the Until operator of CTL with regular expressions.

Our proposal is in line with these latter approaches, but focuses on the translation of CTRL to HMLR, an alternative equational representation of the modal μ-calculus. We were able to obtain an on-the-fly model checker for CTRL by defining and implementing a translation from CTRL to HMLR, and by reusing the verification and diagnostic generation features of the EVALUATOR 3.6 model checker of CADP. This modular architecture allowed us to reduce

the development effort and to take advantage of existing, robust model checking technology.

CTRL is interpreted on Kripke structures, which provide a general description of dynamical systems that implicitly or explicitly underlie many of the existing discrete formalisms used for the modeling of regulatory networks in the cell, such as Boolean networks and their generalizations, Petri nets, and process algebras [30, 56]. In addition, other types of continuous models of regulatory networks, by defining appropriate discrete abstractions, can possibly be mapped to Kripke structures as well. As a consequence, CTRL can be combined with many of the other approaches proposed for the application of formal verification tools to biological regulatory networks [4, 9, 14, 18, 26, 29, 57]. The application of CTRL to biological examples will be shown in Chapter 6.

Although CTRL provides a user-friendlier syntax with respect to existing temporal logics, non-expert users still have difficulties posing relevant and interesting biological questions and formulating them in temporal logic. The next chapter addresses this issue and proposes a solution.

Chapter 4

Patterns

The problem of posing relevant and interesting questions is critical in modeling in general, but even more so in the context of applying formal verification techniques, due to the fact that it is not easy for non-experts to formulate queries in temporal logic (see example in Section 2.5.2). Although one of the objectives of the previously described CTRL language is to provide a simpler syntax to help non-expert users formulate complex biological queries, the formulation of these questions directly into temporal logic remains a difficult task.

The response to this problem proposed by the formal verification community is the use of patterns, that is, high-level query templates that capture recurring questions in a specific application domain and that can be automatically translated to temporal logic [50]. The aim is to provide a set of patterns capable of covering not all possible questions an expert can think of, but rather to simplify the formulation of those that are primary.

During several years the formal verification community has worked on the grouping of recurring questions, like the invariance or the consequence of events, in order to build a base for future questions even between experts in the domain. With the introduction of formal verification techniques in the modeling of biological systems this set of patterns provide a fruitful starting-point to help non-expert users formulate their questions and provide a better and easier integration between these two fields.

In Section 1 we present an overview of the use of patterns in the formal verification and software engineering fields, as well as the types of properties encountered in the systems biology field. Section 2 contains a description of the pattern identification process and the resulting set of patterns and their variants. In Section 3 we describe the translation of the set of patterns into three different temporal logics: CTL, μ-calculus and CTRL. We present some conclusions in Section 4.

4.1 Patterns in formal verification and systems biology

4.1.1 Usage of patterns

The notion of patterns originates in the field of architecture [3] and was introduced in the domain of software engineering as a means to capture expert solutions to recurring problems in program design [59]. This originated an extensive list of patterns (*e.g.*, singleton, abstract factory or proxy) for program design, mostly known in the context of object-oriented programming languages.

In the formal verification domain several attempts have also been made to identify classes of properties that are often used. In the late 70s, one of the first to notice that there exist kinds of properties that one would often want a concurrent program to verify was Lamport [89]. He suggested an informal partition into two categories: *safety* and *liveness*. The first states that something bad never happens, that is, the program never enters an unacceptable state p. The second states that something good eventually does happen, that is, the program eventually enters a desirable state q. The CTL encoding of such properties is $\neg\mathsf{EF}\ p$ and $\mathsf{AG}(p \implies \mathsf{EF}\ q)$, respectively.

In the early 90s, Manna and Pnueli [98] considered three different classes of properties which, according to the authors, were able "to cover the majority of properties one would ever wish to verify". These classes were: *invariance*, stating that all states arising in a computation satisfy p; *response*, stating that every state arising in a computation satisfying p is eventually followed by a state satisfying q; and *precedence*, stating that a state satisfying p initiates a q-interval (*i.e.*, an interval all of whose states satisfy q) which, either runs to the end of the computation, or is terminated by a state satisfying r.

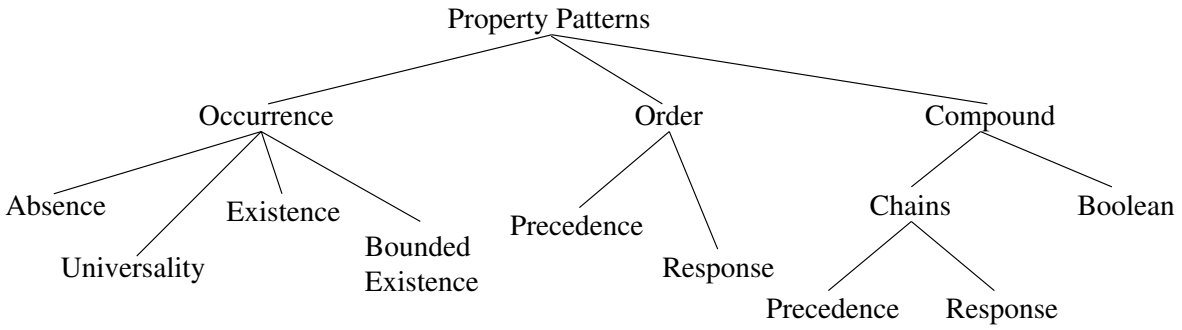


Figure 4.1: Pattern hierarchy proposed by Dwyer *et al.* [50].

In the late 90s, an influential paper by Dwyer *et al.* [50], proposed a *specification pattern system* based on more than 500 examples of property specifications for software specification. This system organizes the patterns according to their semantics into a hierarchy (Figure 4.1). For example, some patterns require events to occur or not occur (*e.g.*, the Absence or Existence

patterns), while other patterns constrain the order of events (*e.g.*, the Response or Precedence patterns). For each of the defined patterns there is an associated *scope*, which is the extent of the program execution over which the pattern must hold. The scope is determined by specifying a starting and an ending state or action depending if the specification formalism is state-based or action-based. There are five basic kinds of scopes: *global* (the entire program execution), *before* (the execution up to a given state/event), *after* (the execution after a given state/event), *between* (any part of the execution from one given state/event to another given state/event) and *after-until* (like between, but the designated part of the execution continues even if the second state/event does not occur).

There are also examples of pattern systems for specific application domains within formal verification. An example is the work by Janssen *et al.* [78], who define a set of patterns for the formal verification of business process models. A large number of business cases were analyzed and lists of relevant questions were extracted and divided into four main categories: *sequence* of activities, *consequence* of activities, *combined occurrence or exclusion* and *required precedence* of activities. For each of the category a pattern in structured natural language was created with a set of available choices, like *an*, *each*, *all*, *ever*, *never* and *always*, acting as quantifiers.

4.1.2 Model checking applications in systems biology

Notwithstanding the existence of the specification pattern system proposed by Dwyer *et al.*, the expertise necessary for its application remains an important obstacle for modelers coming from specific domains like systems biology. A systematic definition of structured biological queries has not received any attention thus far, apart from a list of example queries by Chabrier-Rivier *et al.* [29]. Some of the first applications of model checking in systems biology are presented next, emphasizing on the biological queries used in each study and its representation in natural language. The compilation of these examples serves as a preliminary work for the development of the query patterns in systems biology in the next section.

Chabrier-Rivier *et al.* [29] were one of the first applying model-checking approaches in systems biology. The authors used the symbolic model checker NuSmv [36] to verify properties expressed in Ctl, on models of two biological processes: a qualitative model of mammalian cell cycle control after Kohn's diagrams [83], and a quantitative model of gene expression regulation. For both biological models, a list of biological questions was identified and the authors then explored the possibility of encoding each one of the questions into Ctl. The chosen temporal logic presents some drawbacks, since it cannot answer questions like: What are the stable states? or how long does it take for a molecule to become activated? In Ctl, it is only possible to check whether a specific (or implicitly defined) state is a stable state. Also, the notion of time in temporal logic Ctl is a qualitative notion, providing only an ordering relation between events.

Bernot *et al.* [18] applied model-checking techniques to the analysis of a model of mucus secretion by *Pseudomonas aeruginosa* in lungs affected by cystic fibrosis [66]. The network consists of two genes, *algU* and *mucB*, and two proteins, AlgU and anti-AlgU. AlgU has a positive influence on its own gene and on *mucB*, which in turn has an inhibitory effect on *algU*. The authors focused on the analysis of the bistable behavior of the system. An example of a question encountered in this work is "Does the bacteria regularly produces mucus (the concentration level is repeatedly equal to a)?". Even though it was possible to encode all the questions into CTL, several phrases could be used to represent the same meaning.

Batt *et al.* [11, 14] provided some examples where the qualitative models of genetic regulatory networks are analyzed. An early version of the *E. coli* carbon starvation response network described in Section 6.1.1 of this thesis. An also a generic network consisting of two genes, a and b, and two proteins, A and B. When a gene is expressed, the corresponding protein is synthesized, which, in turn, can regulate the expression of its own and the other gene. Like Bernot *et al.*, the authors focused on the analysis of the bistable behavior of the system, but identified other biologically interesting properties like the reachability of certain equilibrium states and the existence of behaviors satisfying certain constraints on protein concentrations. The authors also stated that the use of a temporal logic more expressive than CTL could facilitate the specification of certain properties, and could be closer to the biological reality.

Calder *et al.* [26] applied model checking techniques to the analysis of a signal transduction pathway. The models were based on high level descriptions of stochastic transition systems, i.e., continuous time Markov chains (CTMCs). Continuous Stochastic Logic (CSL) was used to express and check a variety of temporal queries for both transient and steady state behaviors, using the PRISM [72] model checker. Besides the properties considered in [29], the authors concentrated on further properties which are specific to signaling network models with discretized protein concentrations.

4.2 Patterns of biological queries

4.2.1 Identification of patterns

The difficulty of proposing patterns is to come up with a limited number of query schemas that are sufficiently generic to be applicable in a variety of situations, and at the same time sufficiently concrete to be comprehensible for the non-expert user. Moreover, the overlap between the patterns should be minimal. In order to identify a set of patterns, we started by analyzing a large number of modeling studies in systems biology (starting from the references in [127]), as well as previous applications of model checking and temporal logic (Section 2.4). This bibliographic research allowed us to identify an open-ended list of concepts (Figure 4.2) on the dynamics of genetic, metabolic, and signal transduction networks.

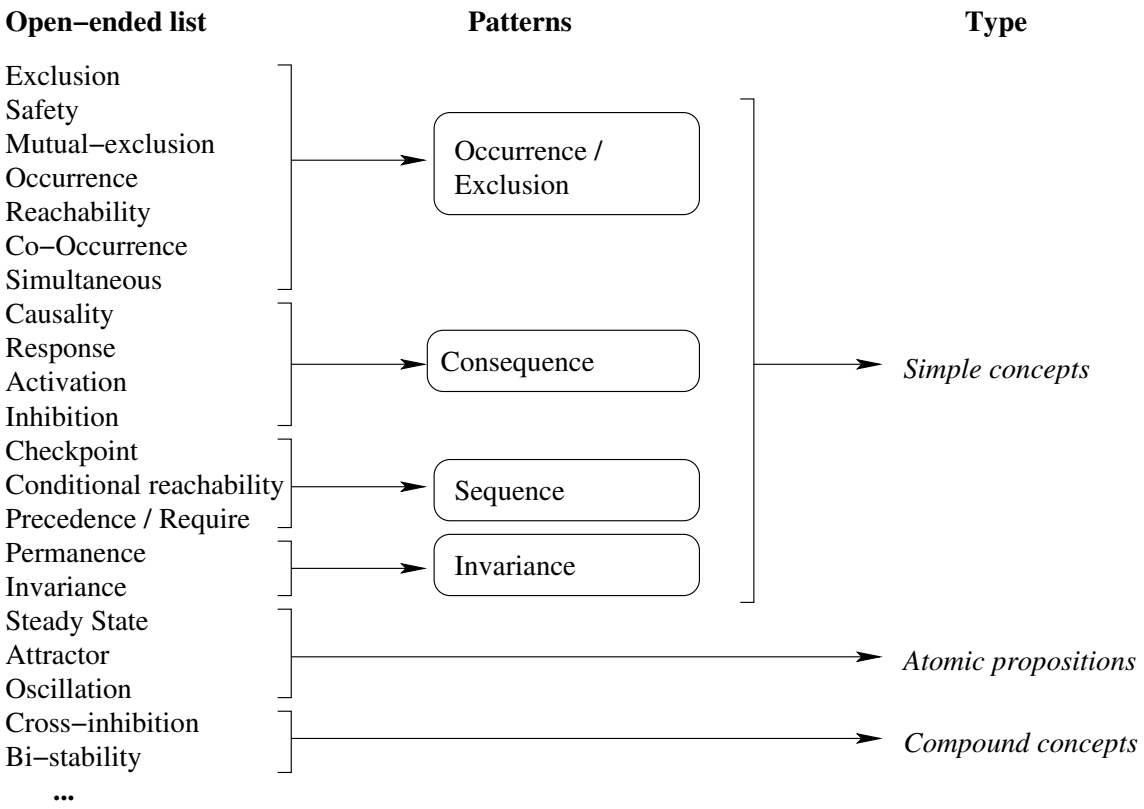


Figure 4.2: Open-ended list of biological concepts used on modeling studies and previous applications of model checking in systems biology.

The identified biological concepts were evaluated and classified into three types: simple concepts, compound concepts and atomic propositions. Simple concepts are concepts that can be obtained using a temporal logic formula and cannot be obtained by combining other simple concepts (*e.g.*, exclusion or causality). Compound concepts are concepts that can be obtained by using a combination of one or more simple concepts (*e.g.*, cross-inhibition). Atomic propositions are concepts that characterize a state of the FSTS rather than a behavior in the FSTS composed by several states (see Section 4.2.2).

Taking into account the natural language meaning of each concept, four main categories of concepts were defined, reaching a compromise between comprehensibility and generality. The categories concerned the *occurrence/exclusion*, *consequence*, *sequence*, and *invariance* of cellular events. For each of these categories, we developed the appropriate pattern in structured natural language, capturing the essence of the question. Also, to discriminate between some concepts within a category, we developed the most relevant variants thus allowing particular biological behaviors to be captured.

4.2.2 Description of patterns

The patterns consist of structured natural language phrases, represented in schematic form, with placeholders for so-called *state descriptors*. A state descriptor is a statement expressing a state property, and takes the form of (a Boolean combination of) atomic propositions. Let ϕ, ψ be state descriptors, then

$$\begin{aligned} \phi, \psi ::= \; & p_1 \in AP \mid p_2 \in AP \mid \ldots \\ & \neg\phi \mid \phi \wedge \psi \mid \phi \Rightarrow \psi \mid \ldots \end{aligned}$$

The state descriptors are interpreted on the FSTS, in the sense that their meaning is formally defined as the set of states $S_1 \subseteq S$ satisfying the state descriptor. In addition to (Boolean combinations of) atomic propositions, the state descriptors may be temporal-logic formulas defined on the atomic propositions. This generalization will be discussed in the conclusion (Section 4.4).

It is often convenient to introduce predefined state descriptors that capture Boolean combinations of atomic propositions that are recurrently used. Some examples of predefined state descriptors that we found useful are the following:

- $Increases_i / Decreases_i$: the concentration of molecular component i increases/decreases in this state;
- $IsSteadyState$: the concentrations of all molecular components are steady in this state;
- $IsOscillatoryState$: this state is part of a set of states containing an oscillation of the concentrations of some molecular components.

Notice that the precise definition of the state descriptors depends on the particular type of FSTS that is used, as the latter determines the set of atomic propositions AP.

Definition 8 (Occurrence/exclusion pattern)

It	is possible	for a state	ϕ	to occur
	is not possible			

This pattern represents the concepts of *occurrence* and its negation, *exclusion*. It will often be used during the development of a model to check for the presence or absence of some property that was experimentally observed. For instance, "It is possible for a state ϕ with a high concentration of protein P_1 to occur". Using this pattern, we can also check for *mutual exclusion*, by using the negation of the pattern in combination with a conjunctive

state descriptor. For instance, "It is not possible for a state ϕ to occur in which there is a high concentration of protein P_1 and protein P_2".

More generally, the *exclusion* pattern captures the safety properties used in the domain of concurrent systems. A safety property (of which mutual exclusion is a typical example) expresses that "something bad never happens" during the execution of the system, in our example a bad state violating mutual exclusion.

Definition 9 (Consequence pattern)

If a state	ϕ	occurs,		
then it is	possibly		followed by a state	ψ
	necessarily			

The *consequence* pattern relates two events separated in time. More precisely, it expresses that if the first state ϕ occurs, then it is possibly or necessarily followed by the occurrence of the second state ψ. If the latter state necessarily follows, then the consequence pattern expresses a form of causal relation. Instances of this pattern are, for instance, "If a state ϕ occurs in which protein P is phosphorylated, then it is possibly followed by a state ψ in which the expression of gene g decreases", or "If a state ϕ occurs in which the concentration of protein P is below 5 μM, then it is necessarily followed by a state ψ in which the expression of gene g is at its basal level".

Definition 10 (Sequence pattern)

A state	ψ	is reachable and			
is	possibly	preceded	at some time	by a state	ϕ
	necessarily		all the time		

The *sequence* pattern represents an ordering relation between two events. It ought not to be confused with the *consequence* pattern, since the conditional occurrence of the second state which characterizes the latter, is absent in the *sequence* pattern. It must be possible to observe both the first and the second state, in that order, for an instance of the *sequence* pattern to be true.

Four variants of the pattern are distinguished, depending on whether the second state ψ follows possibly or necessarily after the first state ϕ, and whether the system is in the first state all the time or only at some time before the occurrence of the second state. Instances of this pattern are "A state ψ in which reactions R_1 and R_2 occur at a high rate is reached after 2 min, and is possibly preceded at some time by a state ϕ in which the transcription factor P is phosphorylated" or "A steady state ψ is reachable and is necessarily preceded all the time by a state ϕ in which nutrient N is absent".

Definition 11 (Invariance pattern)

A state	ϕ	can	persist indefinitely
		must	

The *invariance* pattern is used to check if the system can or must remain indefinitely in a state ϕ. In contrast with the *occurrence/exclusion* pattern, the question is not whether a particular state can be reached, but rather whether a particular state is invariable. Instances of the pattern are "A state ϕ in which reaction R occurs at a high rate can persist indefinitely" and "A state ϕ with a basal expression of gene g must persist indefinitely".

4.3 Translation to temporal logic

By defining a translation into temporal logic of the patterns, the user queries can be automatically cast in a form that allows the verification of the specified property by means of model-checking tools. The set of patterns previously defined are independent of a particular temporal logic, which allows the same high-level specification of a user query to be verified by means of different approaches and tools. It is worth noticing though that some of the patterns we propose have a branching-time nature (*e.g.*, the *consequence* and the *sequence* patterns), and therefore these are not translatable into a linear-time formalism, such as LTL [99].

The following subsections present the translations of the set of patterns into three different temporal logics: CTL, μ-calculus and CTRL. In all translations, formulas are built upon atomic propositions.

4.3.1 Translation into CTL

Natural language query templates	**CTL**
Occurrence/Exclusion pattern	
It **is possible** for a state ϕ to occur	$\mathsf{EF}\ (\phi)$
It **is not possible** for a state ϕ to occur	$\neg\mathsf{EF}\ (\phi)$
Consequence pattern	
If a state ϕ occurs, then it is **possibly** followed by a state ψ	$\mathsf{AG}\ (\phi \Rightarrow \mathsf{EF}\ (\psi))$
If a state ϕ occurs, then it is **necessarily** followed by a state ψ	$\mathsf{AG}\ (\phi \Rightarrow \mathsf{AF}\ (\psi))$
Sequence pattern	
A state ψ is reachable and is **possibly** preceded **at some time** by a state ϕ	$\mathsf{EF}\ (\phi \wedge \mathsf{EF}\ (\psi))$
A state ψ is reachable and is **possibly** preceded **all the time** by a state ϕ	$\mathsf{E}\ (\phi\ \mathsf{U}\ \psi)$
A state ψ is reachable and is **necessarily** preceded **at some time** by a state ϕ	$\mathsf{EF}\ (\psi) \wedge \neg\mathsf{E}\ (\neg\phi\ \mathsf{U}\ \psi)$
A state ψ is reachable and is **necessarily** preceded **all the time** by a state ϕ	$\mathsf{EF}\ (\psi) \wedge \mathsf{AG}\ (\neg\phi \Rightarrow \mathsf{AG}\ (\neg\psi)))$
Invariance pattern	
A state ϕ **can** persist indefinitely	$\mathsf{EG}\ (\phi)$
A state ϕ **must** persist indefinitely	$\mathsf{AG}\ (\phi)$

Table 4.1: Rules for the translation of the patterns into CTL. For each of the four patterns, the translation of all variants is shown.

The first variant of the *occurrence* pattern specifies that it is possible for a given state ϕ to occur, without specifying any conditions or its duration. In CTL it is represented by the potentiality operator EF, that specifies that "there is at least one path leading to a future state satisfying ϕ". The second variant of the *occurrence* pattern specifies the negation of the first. It is often known as a *safety* property and it is represented in CTL by negating the operator EF, specifying that "there is **no** path leading to a future state satisfying ϕ".

The *consequence* pattern specifies a causal relation between two events without any condition regarding the appearance of the first event. This will be specified by the use of the invariance operator AG, which will verify the existence of the causal relation throughout the entire FSTS. The causality is specified through the use of the logical implication. The condition regarding the appearance of the second event generates two variants. The first specifies that it will possibly happen, being represented by the potentiality operator EF. The second specifies that it will necessarily happen, being represented by the inevitability operator AF, specifying that "all the paths lead to future states satisfying ψ".

The *sequence* pattern presents four variants due to duration and branching conditions. The first represents a simple sequence of two events with no time constraints between them, specified through the use of the potentiality operator EF. The second variant states that an event ϕ is always true through the path that leads to the event ψ, being specified with the until operator $\mathsf{E}[\phi\ \mathsf{U}\ \psi]$. The third states that all the attainable occurrences of ψ are preceded by at least one occurrence of ϕ. This is easier to specify in CTL by saying that it is not possible not to have an occurrence ϕ until the occurrences of ψ. The fourth variant specifies that all the occurrences of ψ are always preceded by a constant presence of ϕ. Once again, it is easier to specify that throughout the system it holds that not having ϕ implies not having ψ in the future, using the invariance operator AG. In both the third and the fourth variant the second part of the property is only being verified if there is at least one occurrence of ψ.

The *invariance* pattern specifies that a state persists indefinitely, considering two variants. The first, where a state can persist indefinitely is represented in CTL by the trajectory operator EG, which specifies that "there is at least one path where all its states satisfy ϕ". The second, where a state must persist indefinitely is represented in CTL by the invariance operator AG, which specifies that "all the paths have all their states satisfying ϕ".

4.3.2 Translation into μ-calculus

The *occurrence* pattern represents the possibility of a given state ϕ to occur. In μ-calculus, this concept is represented using the least (μ) fixed point operator together with the possibility ($\Diamond$) modal operator. The use of the logical *or* lets the recursive formula unfold to the next state, if the current state does not yet satisfy ϕ. The second variant is analogous and is obtained by the negation of the whole formula.

Natural language query templates	**μ-calculus**
Occurrence/Exclusion pattern	
It **is possible** for a state ϕ to occur	$\mu X.(\phi \vee \Diamond X)$
It **is not possible** for a state ϕ to occur	$\neg\mu X.(\phi \vee \Diamond X)$
Consequence pattern	
If a state ϕ occurs, then it is **possibly** followed by a state ψ	$\nu X.((\phi \Rightarrow \mu Y.(\psi \vee \Diamond Y)) \wedge \Box X)$
If a state ϕ occurs, then it is **necessarily** followed by a state ψ	$\nu X.((\phi \Rightarrow \mu Y.(\psi \vee \Box Y)) \wedge \Box X)$
Sequence pattern	
A state ψ is reachable and is **possibly** preceded **at some time** by a state ϕ	$\mu X.((\phi \wedge \mu Y.(\psi \vee \Diamond Y)) \vee \Diamond X)$
A state ψ is reachable and is **possibly** preceded **all the time** by a state ϕ	$\mu X.(\psi \vee (\phi \wedge \Diamond X))$
A state ψ is reachable and is **necessarily** preceded **at some time** by a state ϕ	$\mu X.(\psi \vee \Diamond X) \wedge$ $\neg\mu Y.(\psi \vee (\neg\phi \wedge \Diamond Y))$
A state ψ is reachable and is **necessarily** preceded **all the time** by a state ϕ	$\mu X.(\psi \vee \Diamond X) \wedge \nu Y.($ $(\neg\phi \Rightarrow \nu Z.(\neg\psi \wedge \Box Z)) \wedge \Box Y)$
Invariance pattern	
A state ϕ **can** persist indefinitely	$\nu X.(\phi \wedge \Diamond X)$
A state ϕ **must** persist indefinitely	$\nu X.(\phi \wedge \Box X)$

Table 4.2: Rules for the translation of the patterns into μ-calculus. For each of the four patterns, the translation of all variants is shown. We use the version of μ-calculus presented in [87], which is interpreted on classical Kripke structures (Section 2.2.4).

The *consequence* pattern specifies a causal relation between two events throughout the entire FSTS, which is represented by the greatest (ν) fixed point operator together with the necessity ($\Box$) modal operator. There is however the necessity to combine these operators with the logical *and*, in order to verify the current state and the subsequent ones. The possibility variant affecting the second operator uses the same operators used in the *occurrence* pattern. To obtain the necessity variant it suffices to change the modal operator affecting the second event, from possibility ($\Diamond$) to necessity ($\Box$).

The *sequence* pattern presents four variants due to duration and branching conditions. The encoding of the first variant is achieved using one least (μ) fixed point operator together with the possibility ($\Diamond$) modal operator inside another least (μ) fixed point operator together with the possibility ($\Diamond$) modal operator. The second variant uses one least (μ) fixed point operator and a logical *or* to specify that either ψ is true in the current state or ϕ must be true and the same logic is applied to its successors. The third and fourth variants use the equivalent operators in μ-calculus to represent the corresponding CTL formulas.

The encoding of the *invariance* pattern is achieved by the use of the greatest (ν) fixed point operator together with the logical *and* operator. Together these operators encode the persistence concept. To discriminate between both variants, we look at the branching of the successor states, that are encoded by using in the first variant the possibility ($\Diamond$) modal operator and in the second variant the necessity ($\Box$) modal operator.

Natural language query templates	CTRL
Occurrence/Exclusion pattern	
It **is possible** for a state ϕ to occur	$\mathsf{EF}_{\mathsf{true}^*}\ (\phi)$
It **is not possible** for a state ϕ to occur	$\neg\mathsf{EF}_{\mathsf{true}^*}\ (\phi)$
Consequence pattern	
If a state ϕ occurs, then it is **possibly** followed by a state ψ	$\mathsf{AG}_{\mathsf{true}^*}\ (\phi \Rightarrow \mathsf{EF}_{\mathsf{true}^*}\ (\psi))$
If a state ϕ occurs, then it is **necessarily** followed by a state ψ	$\mathsf{AG}_{\mathsf{true}^*}\ (\phi \Rightarrow \mathsf{AF}_{\mathsf{true}^*}\ (\psi))$
Sequence pattern	
A state ψ is reachable and is **possibly** preceded **at some time** by a state ϕ	$\mathsf{EF}_{\mathsf{true}^*}\ (\phi \wedge \mathsf{EF}_{\mathsf{true}^*}\ (\psi))$
A state ψ is reachable and is **possibly** preceded **all the time** by a state ϕ	$\mathsf{EF}_{\phi^*}\ (\psi)$
A state ψ is reachable and is **necessarily** preceded **at some time** by a state ϕ	$\mathsf{EF}\ (\psi) \wedge \neg\mathsf{EF}_{(\neg\phi)^*}\ (\psi)$
A state ψ is reachable and is **necessarily** preceded **all the time** by a state ϕ	$\mathsf{EF}_{\mathsf{true}^*}\ (\psi) \wedge \mathsf{AG}_{\mathsf{true}^*}\ (\neg\phi \Rightarrow \mathsf{AG}_{\mathsf{true}^*}\ (\neg\psi)))$
Invariance pattern	
A state ϕ **can** persist indefinitely	$\mathsf{EG}_{\mathsf{true}^*}\ (\phi)$
A state ϕ **must** persist indefinitely	$\mathsf{AG}_{\mathsf{true}^*}\ (\phi)$

Table 4.3: Rules for the translation of the patterns into CTRL. For each of the four patterns, the translation of all variants is shown.

4.3.3 Translation into CTRL

The encoding of the *occurrence*, *consequence* and *invariance* patterns in CTRL is analogous to the CTL encoding, by using the equivalent form of CTRL operators indexed with the neutral regular expression true^*.

The encoding of the *sequence* pattern differs from the CTL encoding on the second and third variants, exploiting the possibility of using the operator CTRL EF operator together with regular expressions, to express the CTL until U operator. This potentiality operator specifies that at least one of the outgoing paths contains a prefix satisfying the regular expression ρ and lead to a state satisfying φ. By specifying the regular expression ρ as a concatenation of the first argument of the CTL until operator, we get an equivalent expression.

4.4 Conclusions

The widespread adoption of model-checking approaches for the analysis of biological systems is restrained by the difficulty for non-expert users to formulate appropriate questions in temporal logics. Following the work in the formal verification community [50, 98], this chapter presents a formulation of a set of patterns in the form of query templates in structured natural language. The patterns capture a large number of frequently-asked questions by modelers in systems biology, but they are not restricted to a particular type of network, a particular biological system or a particular modeling tool. In addition, translations of the patterns are provided to three different temporal logics, CTL, μ-calculus and CTRL. However, the patterns can be translated to other temporal logics provided that these have enough expressive power to represent the same concept (*e.g.*, being of branching-time nature).

The patterns described in this chapter are globally consistent with those discussed by Dwyer *et al.* [50], but there are differences due to the specific application domain for which our patterns were developed. For instance, the notion of scope used by Dwyer *et al.* [50] is not commonly defined for all the patterns, but implicitly present through the use of specific variants for each pattern. Also, we have not explicitly included patterns that can be obtained by the recursive application of other patterns, such as the *chain response* pattern defined in Dwyer *et al.* [50]. While patterns have not been used for the querying of cellular interaction networks thus far, some papers list example questions. It is reassuring to observe that all questions listed by Chabrier-Rivier *et al.* [29] can be expressed by means of the patterns in Section 4.2.2.

An obvious generalization of the patterns described in this chapter, already briefly mentioned, would be to allow state descriptors that are formulas in temporal logic. For instance, instead of using atomic propositions to label states belonging to a (terminal) cycle in the FSTS, which requires the preliminary detection of strongly connected components in the state transition graph, we could use temporal logic formulas [14]. The introduction of temporal logic formulas as state descriptors makes the patterns more general, but also potentially more complicated to formulate and dependent on a particular temporal logic. A compromise trading some expressive power for user-friendliness would be to restrict the possible temporal logic formulas to simple forms and introduce these as predefined state descriptors (Section 4.2.2). This is consistent with the main idea underlying the use of patterns, namely that they cannot be expected to cover all possible queries, but rather should allow users to formulate frequent questions without worrying about their translation to temporal logic.

Chapter 5

Implementation of a service-oriented architecture

Despite the fact that several examples of the application of model checking to the analysis of biological regulatory networks have been published in the literature (*e.g.*, [4, 8, 9, 14, 15, 18, 21, 26, 29, 57, 64, 123]), two major obstacles still prevent modelers in systems biology from drawing maximal benefit from formal verification tools.

First, the formulation of biological questions in temporal logic (shown in Chapter 4) and the interpretation of the verification results is far from obvious, especially for non-expert users who are not used to this kind of reasoning. Second, most of the existing modeling and simulation tools are not capable of applying model-checking techniques in a transparent way. In particular, they do not hide from the user the technical details of the installation of the model checker, the export in a suitable format of the model and the query, the call of the model checker, and the import of the results produced by the model checker (the true/false verdict and witnesses/counterexamples). In other words, what is missing is a framework that tightly integrates modeling and simulation tools with formal verification tools, on both the conceptual and the implementational level.

In order to address these issues, we propose a service-oriented architecture (SOA) [54] for the integrated modeling and formal verification of genetic regulatory networks, which reuses existing technology as much as possible. The architecture connects modeling and simulation clients to a formal verification server, *via* an intermediate request manager. In particular, the client can perform verification requests through the web, which the request manager dispatches to an appropriate formal verification server. When the formal verification server has answered the request, the results are sent back to the modeling and simulation client for display and further analysis in the graphical user interface of the tool. The interactions of the client with the remote web server are handled by a verification module assisting the specification of biological queries through a property editor, either by directly choosing the appropriate temporal logic operators or by using a tailored set of query patterns [109].

The first section provides a description of the architecture and presents the used modeling and simulation tool GNA [45]. Section 2 describes the verification module, showing the pattern-based property editor, the client-side plugins and the connection with the request manager through a running example. Section 3 presents the request manager, in particular the queue and authentication manager and the load balancer. Section 4 presents the formal verification server and the server-side plugins. Section 5 describes the possibility of integration of new formal verification tools, as well as new modeling tools into the architecture. Section 6 provides some concluding remarks about its usefulness.

5.1 Service-oriented architecture

Figure 5.1 shows the overall architecture of the developed system, specifying its three components: the modeling and simulation tool with its verification module, the request manager, and the formal verification server. These three components have been implemented in Java 1.5 and their web-service interface is based on Apache Axis 1.3 (`http://ws.apache.org/axis/`).

The implementation followed two main principles: a service-oriented architecture and the use of plugins. A service-oriented architecture is particularly well suited for our purpose. The formal verification service is remotely executed through the web and is implemented using standard protocols and languages like TCP/IP, SOAP and XML. A user of a modeling tool wishing to perform a verification request does not need to install a model checker or other formal verification tool locally on his or her machine. The use of plugins provides a flexible and extensible way to abstract a particular formal verification tool. It allows one to apply the tool without worrying about the details of its implementation.

5.1.1 Modeling and simulation tool

The service-oriented architecture is accessible for users of version 7.0 of the qualitative modeling and simulation tool Genetic Network Analyzer (GNA)[1]. GNA uses a class of piecewise-linear (PL) differential equations, originally introduced by Glass and Kauffman [65], providing a coarse-grained picture of the dynamics of genetic regulatory networks, well-adapted to the current lack of quantitative information on many networks of interest. The models associate a protein concentration variable to each of the genes in the network, and capture the switch-like character of gene regulation by means of step functions that change their value at threshold concentrations of regulatory proteins. The advantage of using PL models is that the qualitative dynamics of the high-dimensional systems are relatively simple to analyze, using inequality constraints on the parameters rather than exact numerical values [13, 14]. This makes the PL models a valuable tool for the analysis of genetic regulatory

[1] GNA is distributed by Genostar (http://www.genostar.com/). Free license for non-commercial academic users granted upon request on the GNA home page (http://ibis.inrialpes.fr/article122.html).

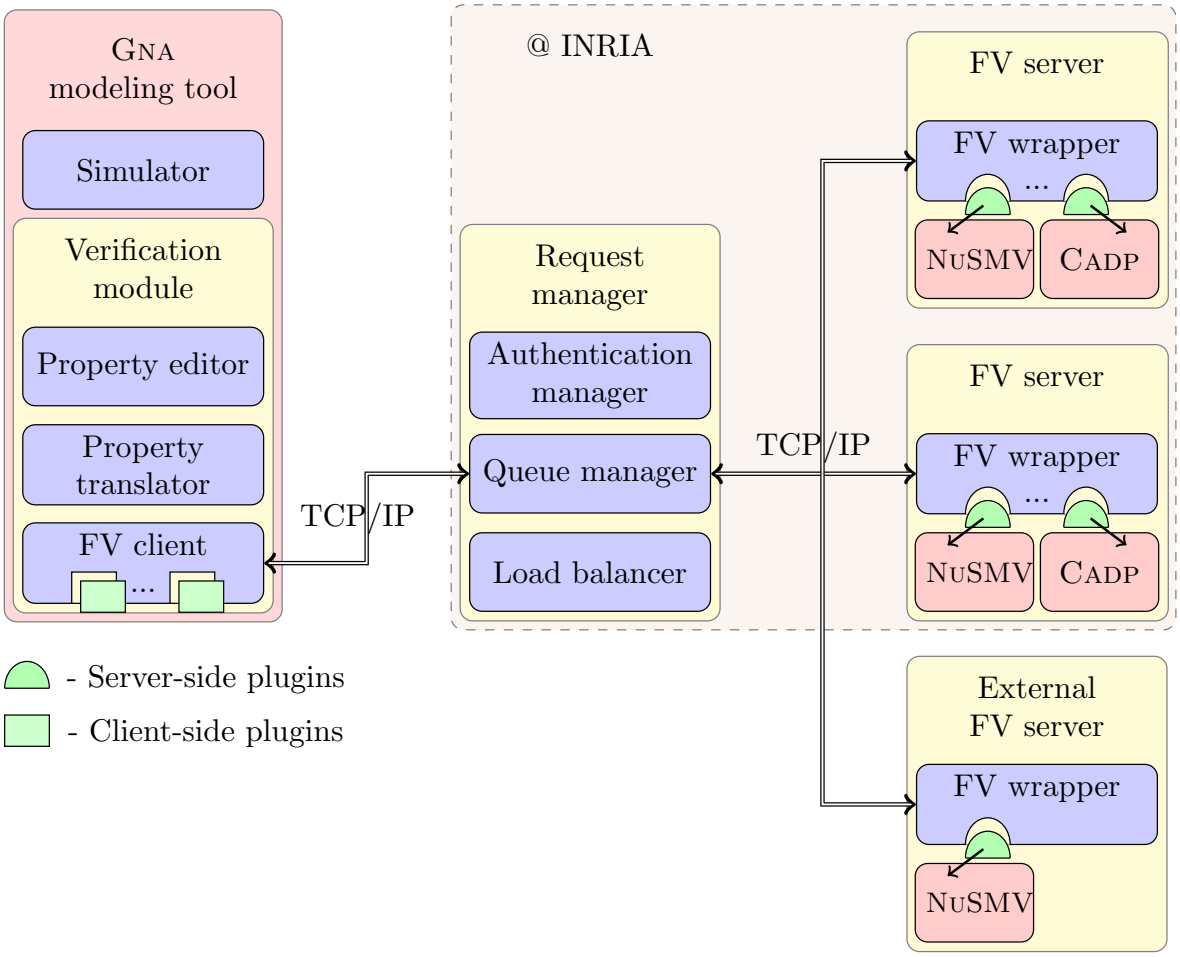


Figure 5.1: Service-oriented architecture for the integration of tools for the modeling and simulation of genetic regulatory networks with formal verification (FV) tools. In particular, the architecture has been implemented to connect GNA with the model checkers NUSMV and CADP. GNA is extended with a verification module responsible for the transformation of the model and properties into a format specific to a formal verification tool, and for the communication with the other components of the service-oriented architecture.

networks in the absence of quantitative information on the parameter values. The graphical user interface of GNA supports the modeler in building step-by-step a PL model of the network under study (see the tutorial available at the GNA web site, for details and examples).

GNA computes discrete abstractions of the continuous dynamics of the PL models, resulting in a finite-state transition system (FSTS) [40] (see Section 2.1 for a formal definition). Each of its states corresponds to a hyperrectangular region in the concentration space, defined by the thresholds of the concentration variables. And each of its transitions corresponds to a solution trajectory entering one region from another [13].

GNA allows the user to visualize the FSTS, *i.e.*, to display the corresponding state transi-

tion graph, and analyze the atomic propositions characterizing the states. For large graphs visual inspection quickly becomes infeasible and formal verification tools are needed. Previous versions of GNA supported the export of the FSTS to text files accepted by several model checkers [12, 14]. Version 7.0 extends GNA with a verification module that integrates the tool into the service-oriented architecture.

5.2 Verification module

The verification module consists of three components: a pattern-based property editor, a property translator, and a formal verification client (Figure 5.1).

5.2.1 Pattern-based property editor and translator

The problem of posing relevant and interesting questions is critical in modeling in general, but even more so in the context of applying formal verification methods, due to the fact that is not easy for non-experts to formulate queries in temporal logic. The pattern-based property editor is a user interface implementing the set of patterns defined in Chapter 4. It allows the specification of biologically-relevant properties in the form of temporal logic formulas. This specification can be achieved in two distinct ways: for common biological properties through the use of a pattern system, and for more specific or complex properties through the use of a text editor of temporal logic formulas.

Figure 5.2 shows the pattern-based property editor of GNA. It presents the four different types of patterns as templates to be completed by the user. The completion of the templates requires the modeler to have previously defined atomic propositions, each of which describes characteristics of a state of the network, such as an increasing or decreasing protein concentration, a steady state, or a protein concentration above a certain threshold. When a pattern has been specified, it is automatically translated into Computation Tree Logic (CTL) [40]. The pattern-based property editor and translator were also made available as a stand-alone Java application under the name Procrustes[2]. An application programming interface (API) is also provided, so that the patterns can be integrated into other modeling tools that wish to implement the encoding of biological properties into temporal logic formulas (CTL, CTRL and μ-calculus are currently supported).

More complex biological properties can be directly specified in the CTRL language (see Chapter 3 for a detailed description) [102]. The text editor allows the modeler to specify any temporal logic formula by freely combining the set of CTL and CTRL operators with propositional logic operators and the user-defined atomic propositions. The temporal logic properties can be stored for later use with the GNA model in a single project file.

[2]Procrustes: pattern-based property editor is publicly available (http://ibis.inrialpes.fr/article938.html).

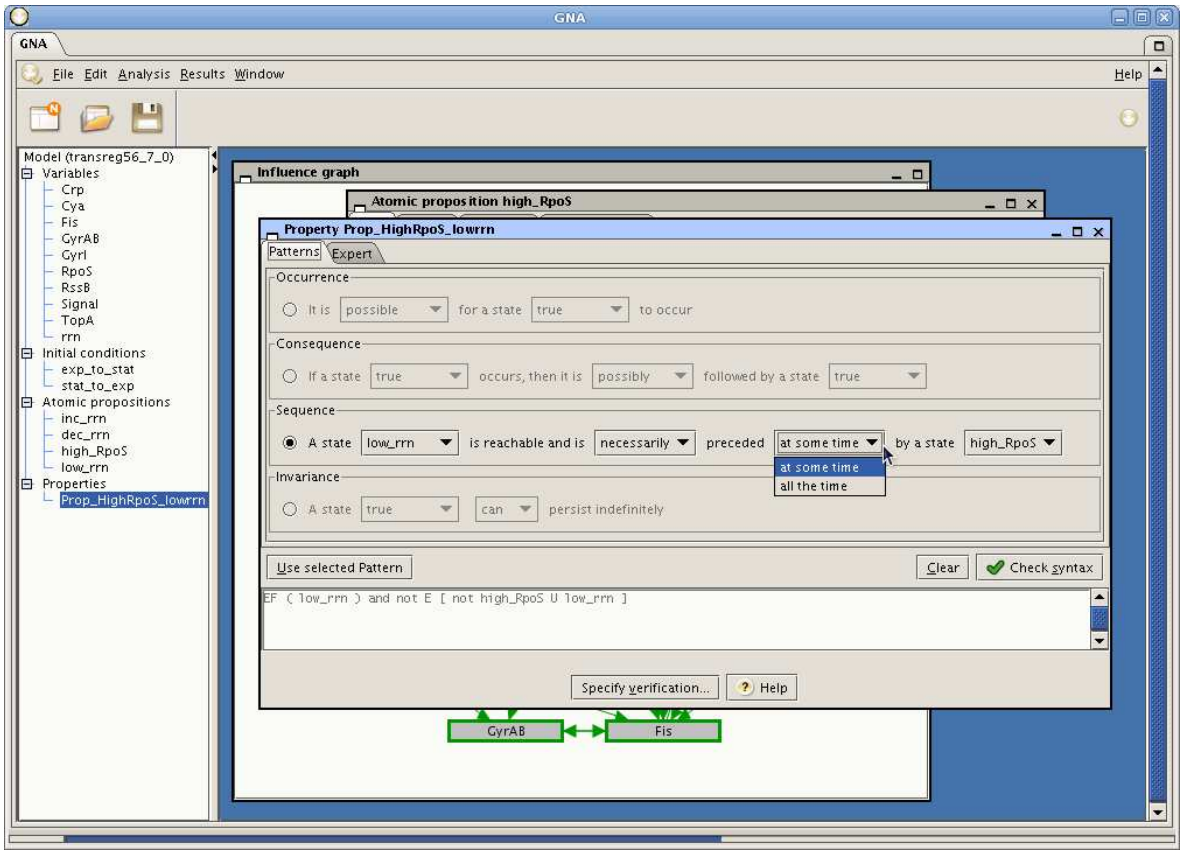


Figure 5.2: Graphical user interface for the specification of biological properties. The modeler can use a pattern-based property editor for frequently-asked questions, and a text editor for the specification of more complex biological properties (expert mode).

In order to illustrate the use of the developed architecture, we consider as a running example a scenario of the network of global regulators controlling the carbon starvation response in the enterobacterium *Escherichia coli* (see Section 6.1 for a detailed description). Given the important role of RpoS for the survival of the cell, we are interested in verifying if the entry into stationary phase upon carbon starvation is always preceded by the accumulation of RpoS in the cell.

The first step in answering this question using the formal verification module of GNA consists in identifying elements of the question that refer to the state of the biological system and in stating these as atomic propositions. We represent the entry into stationary phase of the system by a low level of stable RNAs encoded by the *rrn* operons. This characteristic is specified using the property editor, where we create an atomic proposition named *low_rrn* (Figure 5.3), restricting the concentration values for the variable *rrn* to those below its (single) threshold. We also introduce an atomic proposition *high_RpoS*, representing the accumulation of RpoS to a value above its threshold *t_RpoS*.

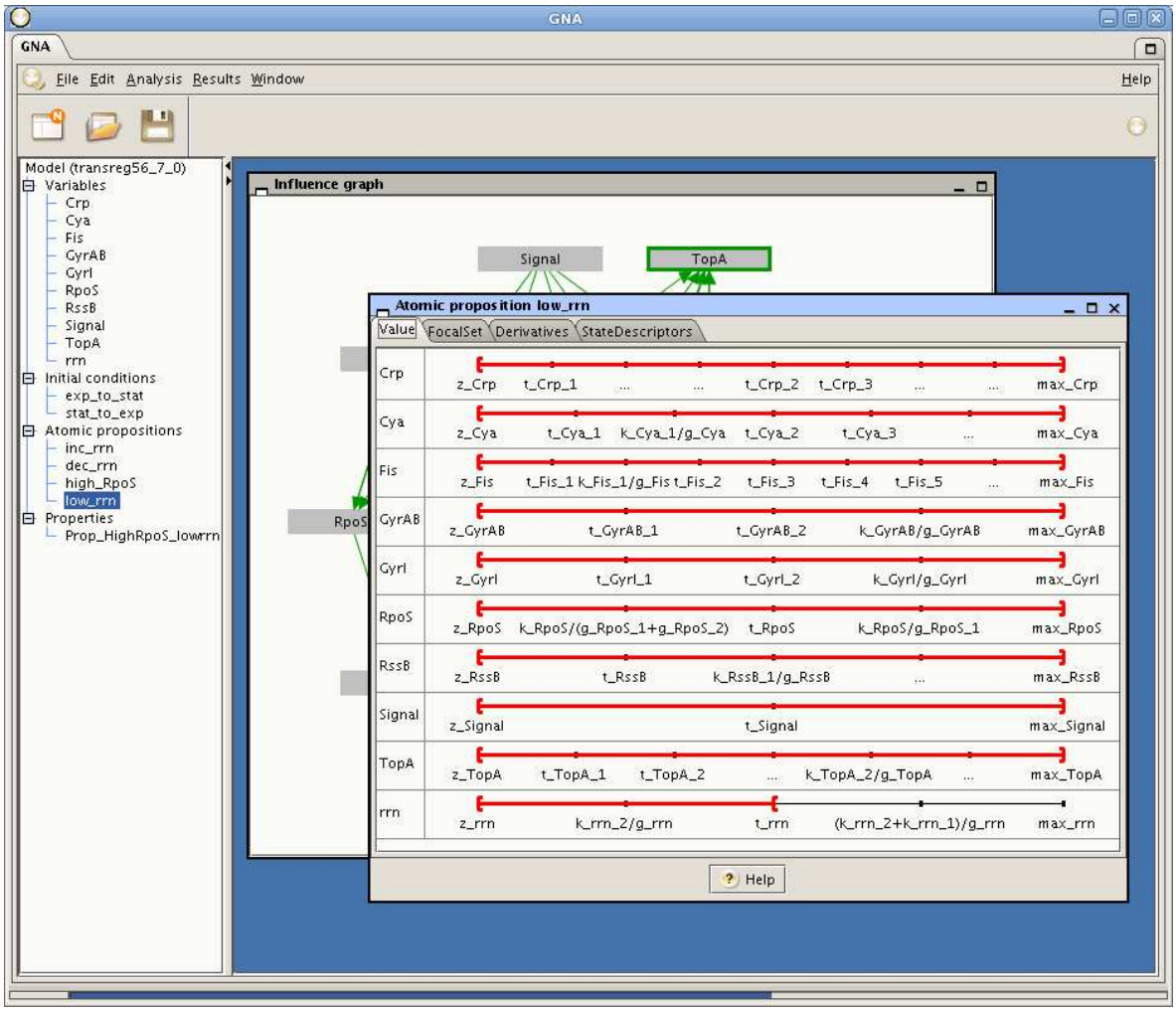


Figure 5.3: Atomic proposition specification window, where atomic propositions are defined in terms of restrictions applied to a state (*e.g.*, restrictions on concentration values, focal sets, derivatives, and other state descriptors). In this case, the value of the concentration is restricted to lie below the threshold *t_rrn*.

The second step is the formulation of the biological property using the pattern-based property editor and translator. We choose the *sequence* pattern to account for the temporal ordering of the two states: stationary phase and high expression of RpoS. The sequence pattern is instantiated by selecting the previously defined atomic propositions (Figure 5.2):

A state	*low_rrn*	is reachable and is		
necessarily	preceded	*at some time*	by a state	*high_RpoS*

Once the pattern is fully instantiated, it is automatically translated into the corresponding

CTL formula: $\mathsf{EF}\ (low\_rrn) \wedge \neg \mathsf{E}\ (\neg high\_RpoS\ \mathsf{U}\ low\_rrn)$.

5.2.2 Formal verification client and client-side plugins

The formal verification client is the component that enables GNA to communicate with the request manager. It thus gives the user an easy access to the formal verification technology without having to locally install a tool or worrying about how to get it to work. To perform a verification request, the modeler needs to choose which tool to use, which property to verify, *etc.* These choices may be guided by the estimation of the model size (*e.g.*, for large regulatory networks containing dozens of genes, symbolic model checking is likely to scale up better than explicit-state model checking) or by the nature of the properties to be verified (*e.g.*, linear-time or branching-time, with/without regular expressions, *etc.*).

The FSTS on which the property is to be verified can be defined explicitly or implicitly. In the former case, the FSTS is completely generated by the simulation module of GNA, while in the latter case it is given by the set of initial states and a function that computes the successors of any given state. The formal verification client performs a request by sending the implicit or explicit description of the FSTS through the web and waiting for the result. The implicit definition has the advantage of considerably reducing the size of the specification of the FSTS, and thus limiting the size of the files transmitted and the response delays. This may be critical for large FSTSs. The verification result is composed of a true (false) verdict supported by a witness (counterexample). The witness or counterexample consists of a sequence of states in the FSTS, displayed in the graphical user interface of GNA.

In order to make the verification module of the modeling tool independent of a specific formal verification tool, we have developed a plugin system. Currently, a plugin for the model checker NUSMV is available, while a beta version for CADP has been completed. All data transformations specific to a particular model checker are taken in charge by the corresponding plugin, thus leaving the service-oriented architecture free to manage generic verification requests. Each plugin has a client-side and a server-side (see Figure 5.1). The client-side plugin has the responsibility of translating the FSTS and the property into a format accepted by the corresponding formal verification tool, while the server-side plugin is in charge of receiving the translated FSTS and property, feeding them into the formal verification tool executable, and parsing the results returned by the tool.

At the present time, the model checkers integrated in the architecture are invoked using the default parameters. More elaborate choices could be partially automated by incorporating into the plugin some knowledge of the verification method and the underlying algorithms.

Continuing with the considered running example, we can now pass to the verification stage by configuring the verification request in the verification window (Figure 5.4). First, we choose the name and version of the model checker plugin to be used (version 1.0 of the NUSMV implicit plugin was used in the example). Second, we specify the initial conditions.

The resulting implicit FSTS represents the transition from exponential phase to stationary phase, starting from initial conditions corresponding to carbon depletion. We then run the verification request, which is sent to the Request manager component, which will be described in the next section.

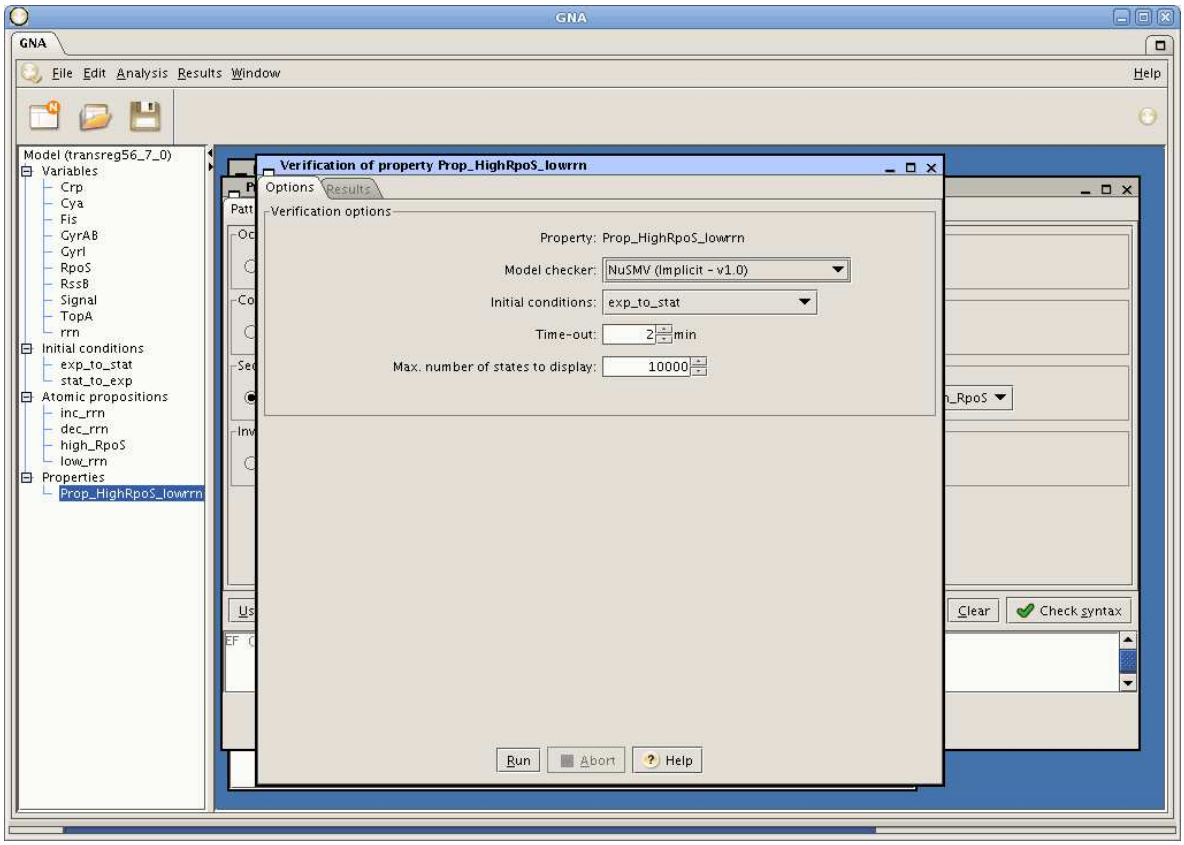


Figure 5.4: Configuration of a verification request by specifying the model checker plugin to be used and, if the plugin supports an implicit representation of the FSTS, the initial conditions for the qualitative simulation of the network.

5.3 Request manager

The request manager is a component of the service-oriented architecture with a public address (`http://java1.inrialpes.fr`), acting as an intermediary service that ensures the communication between all the modeling tools and formal verification servers.

5.3.1 Queue and authentication manager

In order to keep track of the state of all verification requests and the available formal verification servers, a queue and authentication manager have been implemented. Upon each

verification request the authentication manager, together with the server-side plugin, checks for the credentials of the request. If successful, the queue manager registers the request in the queue, checks for an available formal verification server, and hands over the request. The queue manager will continue to poll the formal verification server for a response until one of three events happens: the verification has completed, the user has aborted the verification request, or a timeout has occurred. The verification result (verdict and witness/counterexample) is then returned to the user.

To ensure the service security, each authenticated request registered in the queue, generates an Universally Unique Identifier (UUID) that is returned to the client, so that only this client is able to retrieve the verification result. Furthermore, when the result is retrieved, both the request manager and the formal verification server, that handled the request, delete the model and temporal logic formula, leaving no traces of the request in the server.

5.3.2 Load balancer

The service-oriented architecture has been designed to support several formal verification servers. The address of every server, as well as all the types of model checkers and other formal verification tools locally installed on each of the servers, are registered in the request manager. Upon a verification request, the load balancer chooses an idle formal verification server with the required tool and server-side plugin installed. When all formal verification servers are busy, the load balancer waits until one becomes idle.

5.4 Formal verification server

A formal verification server has the responsibility of verifying properties submitted by the request manager. One or several formal verification tools can be installed on a server provided that the corresponding server-side plugins are also installed on this server.

5.4.1 Formal verification server and server-side plugins

The formal verification server contains the web-service interface, which is responsible for receiving the requests from the request manager, the choice of the corresponding server-side plugin, and the construction of the verification result to be returned.

Each plugin specific to a formal verification tool has an authentication module which responds to the authentication requests made by the request manager. In addition, upon a verification request, the plugin pre-processes the model description and the property in order to transform them into the format accepted by the formal verification tool, and calls the latter with the appropriate parameters. When the formal verification tool finishes the verification of the request, it produces the verdict as well as the corresponding witness (or counterexample).

Since this witness has a format specific to a particular formal verification tool, it is up to the plugin to parse the results and perform the necessary data transformations to a common format that is sent back to the modeling tool. The required transformations depend on whether the request involves a FSTS of an implicit or explicit type. In the explicit case, the witness is simply a subgraph of the FSTS sent to the formal verification tool, whereas in the implicit case the state information needs to be reconstructed from the output of the tool.

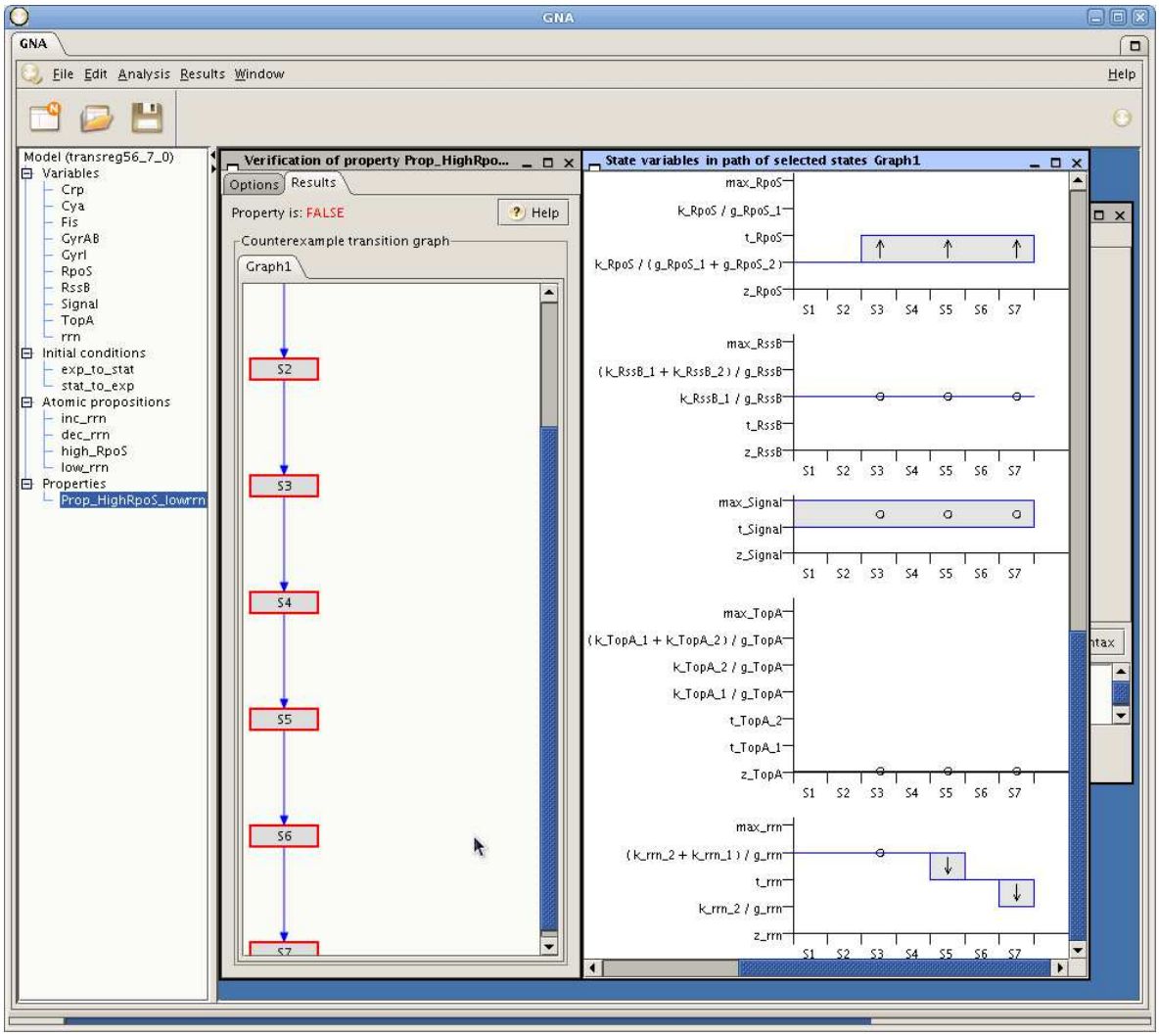


Figure 5.5: Result of the verification of a biological property, consisting of a false verdict and the corresponding counterexample composed of a subgraph of the FSTS (see left panel). The qualitative evolution of the concentration variables of the selected states of the counterexample can also be visualized (see right panel).

Concluding the running example, the model checker returns false after 4 seconds. It means that the entry into stationary phase is not always preceded by the accumulation of RpoS in

the cell. The counterexample is presented to the user as shown in the left panel of Figure 5.5. It consists of a subgraph of the initial Fsts, starting from the specified initial state and ending in a state where the property fails. By selecting a path in this subgraph, Gna allows the qualitative changes in the concentration of all the variables to be displayed (right panel of Figure 5.5). Looking at the evolution of the variables we immediately observe that there is (at least) one sequence of states leading to a low expression level of the *rrn* operons without having previously passed through a state with a high concentration of RpoS. This illustrates the negative verification result, and witnesses that the downregulation of the stable RNAs does not require the previous accumulation of RpoS.

5.5 Integration of new tools

The architecture is generic and modular, but we develop it here in the context of one particular modeling and simulation tool (Gna [45]) and two different model checkers (NuSmv [36] and Cadp [62]). This modularity permits the delegation to plugins of all operations that are specific to a particular tool, allowing a flexible integration of new tools. Detailed information on the integration of new tools with the developed architecture can be obtained by contacting directly the IBIS team[3].

5.5.1 Integration of new modeling tools

A first generalization of this architecture would be to extend it to other modeling and simulation tools. A variety of tools have been used in combination with model checkers, such as GINsim [110], Ina [125], Biocham [27], Gna [45] or Rovergene [15], based on formalisms like Boolean and other logical models [95, 124, 134], Petri nets [31, 64, 82] and ordinary differential equations [20, 44]. Developers wanting to integrate their modeling tools with the architecture can do so by developing a verification module. This verification module will be responsible for several tasks. Firstly, the specification of biological properties and their translation into temporal logic formulas. This can be easily obtained by integrating the publicly available pattern-based property editor Procrustes. Next, both the temporal logic formulas and the model must be given to the client-side plugin that will perform the necessary data transformations to be recognized by the corresponding formal verification tool installed on a remote formal verification server. Thirdly, the developer must implement all the web-service methods (specified in the public address http://java1.inrialpes.fr/) to correctly submit verification requests to the request manager, as well as to receive the corresponding verification result. Finally, the verification module must implement a graphical user interface to correctly display the verification results.

[3]IBIS team website (http://ibis.inrialpes.fr/)

5.5.2 Integration of new formal verification tools

A second generalization of this architecture would be to integrate other formal verification tools into the architecture. This possibility is anticipated through the use of a plugin system, where each plugin contains all data transformations and operations specific to a particular formal verification tool. This simplifies the integration of a new tool to the creation of the corresponding plugin. Two plugins have been developed until now: one for NuSmv (released with the Gna distribution) and one for Cadp (beta version completed). Developers wanting to integrate plugins for different model checkers or other formal verification tools can do so through the following main steps: the development of a client-side plugin, the development of a server-side plugin, and the installation of the server-side plugin on a server on which the new tool is running.

The client-side plugin takes a .jar file that must be placed in the *plugins* directory of Gna, allowing the modeling tool to export the Fsts to a file that can be read by the new formal verification tool. Gna dynamically recognizes the available client-side plugins, using the Java Plugin Framework technology. The development of a server-side plugin results in a Java class that needs to be copied in an appropriate directory of the wrapper on the formal verification server. The latter server must register its web service connection parameters in the request manager, so as to enable the latter to dispatch the requests to the correct formal verification server.

5.6 Conclusions

We have proposed a generic and modular service-oriented architecture to integrate the modeling of genetic regulatory networks with existing formal verification tools. Currently, the service-oriented architecture connects the Gna modeling tool, extended with a formal verification module, with the NuSmv and Cadp model checkers. We have given a detailed description of the existing components and motivated our implementation decisions. Additionally, we have illustrated the use of this architecture with the analysis of the complex network of global regulators involved in the carbon starvation response in *E. coli*, which will be further discussed in the next chapter. Gna is freely available for non-profit academic research, while the main component of the formal verification module, the pattern-based property editor and translator, is also available separately.

Formal verification methods have historically been used for the verification of hardware and software systems. Some of the existing model checkers, such as Prism [72] and NuSmv [36], have recently been applied to the verification of biological systems. Prism verifies properties specified in Continuous time Stochastic Logic (CSL) and has been used to perform quantitative analysis [26] of the ERK intracellular signaling pathway model [35]. NuSmv has been used for the analysis of biological models like the carbon starvation response

in *E. coli* [14], the cell-cycle control in *C. crescentus* [123], the mucus production in *P. aeruginosa* [18], and the mammalian cell-cycle control [29]. In most cases, the biological models are built using modeling tools that are not connected to model checkers. Some modeling tools like GINsim [110] and previous versions of Gna [12] are capable of exporting the model in an implicit or explicit format accepted by the model checker and the entire analysis is carried out in the model checking environment, without any feedback to the modeling tool. An exception is the modeling tool Biocham [27], which integrates the model checker NuSmv and allows for a more flexible iterative modeling and verification approach.

We carried further the integration of the modeling and formal verification of biological networks, by proposing a service-oriented architecture that presents several advantages. First of all, the proposed connection between modeling and verification tools is completely transparent for the modeler and platform-independent. It requires web access but this is becoming less and less of a constraint in the current age of pervasive internet use. Second, the web-service based integration of the tools coming from different domains makes it possible to exploit the strong points of each. On the modeling side, the graphical user interfaces present the properties to be verified and the verification results in a way accessible to the modeler. For instance, the specification of biological properties by means of query patterns (see Chapter 4) [109] does not require prior knowledge of any specific temporal logic. On the verification side, the latest developments of state-of-the-art model checkers can be immediately integrated. Third, the plugin system provides a modular way to add new formal verification methods without having to develop a new version of the modeling tool. The upgrade to future releases of a formal verification tool can also be performed through a simple plugin update.

The architecture has been implemented in the context of Gna, but generalizations to other modeling and simulation tools is obviously possible and facilitated by the modular structure. The integration of such tools into the architecture requires them to implement a verification module responsible for the specification of biological properties, the call of plugins for specific formal verification tools and the exchange of verification requests with the request manager. However, this implementation work is facilitated by the availability of the pattern-based property editor as a stand-alone Java application. In addition, the development of new plugins for tools based on model formalisms that can be mapped to Fstss, explicitly or implicitly, are conveniently designed after plugins already available for Gna.

The modular infrastructure that we proposed is capable of connecting modeling and formal verification tools. In combination with graphical user interfaces capable of presenting data in a form accessible to modelers, we expect this to lower the obstacles to the use of formal verification technology in the analysis of biological regulatory networks.

Part III

Applications and discussion

Chapter 6

Analysis of genetic regulatory networks in *E. coli* and *S. cerevisiae*

After the definition and implementation of the methods developed in the context of this PhD thesis, this chapter shows their practical application. It illustrates the analysis of two biological systems from two different model organisms, the bacterium *Escherichia coli* and the eukaryote *Saccharomyces cerevisiae*. The analysis of these two systems allowed a better understanding of the biological processes under study.

The first model concerns the response of the global regulators of transcription of *E. coli* cells upon carbon starvation conditions. We first start by providing a description of the biological process under study. Then, a piecewise-linear model is presented which tries to predict the response of the *E. coli* cells to carbon starvation. Finally, an analysis and verification of this model is performed, using the previously described methods.

The second model concerns the *FLR1* response to the stress induced by the fungicide mancozeb in *S. cerevisiae*. Analogously, we start by providing a description of the biological process under study. Then, we present a piecewise-linear model which predicts the *FLR1* response to the presence of mancozeb. Finally, we perform the analysis and verification of the model.

These two models are defined using the GNA modeling and simulation tool (Section 5.1.1) to generate their possible behaviors. These behaviors are exported by GNA as FSTSs, which are then verified for the occurrence of the observed biological behaviors with the help of the previously described formal verification architecture (Chapter 5). These behaviors can be represented in two distinct ways. Either by simulating all the reachable states and representing them explicitly through a FSTS, or through an implicit representation where the transitions between states are only explored at the model checking stage.

Each of the properties to be verified can be specified either by using the pattern-based property editor (Chapter 4) or by directly using the CTRL language (Chapter 3) in the case of complex biological properties with the need for more expressive power.

The model checkers considered in the analysis and verification of these two biological models were NUSMV and EVALUATOR from CADP toolbox. For NUSMV both the implicit and explicit FSTS representation were used to test properties of the model. For EVALUATOR only the explicit FSTS representation was considered, because the implicit representation is not implemented yet.

The documentation of the verification procedure is enriched with screen-shots of GNA's graphical user interface, which can be found in Appendix B.

6.1 Analysis of carbon starvation response in *E. coli*

6.1.1 Carbon starvation response in *E. coli*

In order to survive, *E. coli* cells constantly have to adapt their functioning to the availability of carbon sources, essential for growth. Under favorable environmental conditions, bacterial cells quickly grow and divide, leading to an exponential increase of their biomass, called *exponential phase*. Upon a variety of stress conditions, like the depletion of carbon sources, the bacteria abandon exponential phase and enter a state in which cells stop dividing, capitalizing upon the few available resources to maintain the basic metabolic functions necessary for survival. This so-called *stationary phase* is rapidly reversed and fast growth restored once the environmental conditions become favorable again [74].

Glucose is the preferred carbon source of *E. coli*. The adaptation of the bacteria to the depletion of glucose from the growth medium is under the control of a large and complex network involving multiple levels of regulation, from metabolic fluxes and enzyme activity to gene regulation [67, 69, 85]. The role of the metabolic and signalling networks in the adaptation of *E. coli* to carbon source starvation have been extensively studied (e.g., [19, 32, 117]), but much less has been done at the level of gene expression. For this system[1], we focus on the role of the global regulators of transcription, such as CRP, Fis, DNA supercoiling, and RpoS. These global regulators form the backbone of the network coordinating the long-term response of *E. coli* cells to starvation conditions (Figure 6.1). The transcription factors respond directly or indirectly to glucose depletion, by controlling in a combinatorial fashion the expression of a large number of genes involved in cellular adaptation and survival. In addition, they control each other's expression, thus giving rise to a complex regulatory network. It is not well understood how the interactions between the global regulators coordinate the cascades of molecular events driving the growth arrest of *E. coli* cells starved for glucose.

[1] In this subsection, the notation used for the gene and protein names are those consistent with the community working with the *E. coli*: use of italic minuscules for gene names (*i.e.*, *crp*, *fis* and *cya*) and non-italic for protein names (*i.e.*, CRP, Fis and Cya).

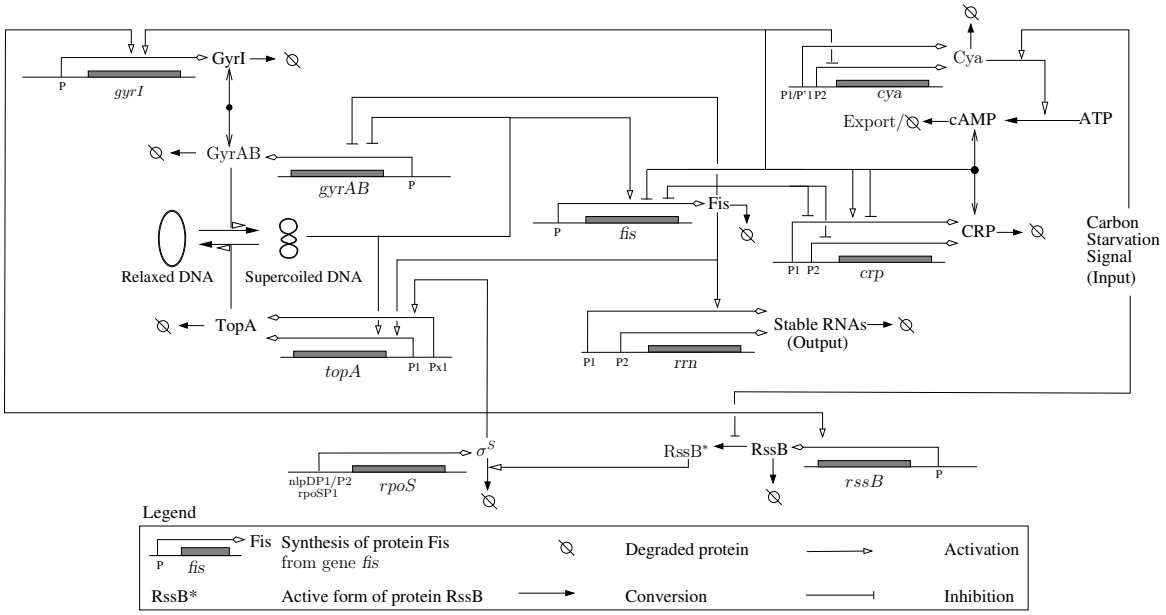


Figure 6.1: Network of key genes, proteins and regulatory interactions involved in the carbon starvation response network in *E. coli* [109, 118]. It includes well-known pleiotropic transcription regulators, like the histone-like protein Fis, the catabolic repressor cAMP·CRP (resulting from the expression of genes *crp* and *cya*, and the activation of Cya by carbon depletion), and the general stress response factor RpoS or σ^S (whose stability is regulated by RssB). Changes in DNA topology and its dependence on the relative expression level of the genes *gyrA*, *gyrB*, *gyrI*, and *topA* are also considered, as the three-dimensional structure of DNA modulates the transcription of a large number of genes. Finally, stable RNAs expressed from the *rrn* operons are considered as their amount provides a reliable indicator of the growth rate of the cell, being high during an exponential phase and low during a stationary phase.

6.1.2 Model of carbon starvation response in *E. coli*

Due to the lack of biological data, it is not possible to build a quantitative model of this network. There is, however, enough data available in the literature to build a qualitative model, in order to analyze the dynamic behavior of the regulatory network. A piecewise-linear model (Section 5.1.1) for the carbon starvation network was therefore built. The qualitative dynamics of the network are described by nine coupled piecewise-linear differential equations, and fifty inequality constraints on the parameter values (Table 6.1) [109, 118]. The qualitative analysis of a network of this size and complexity generates huge Kripke structures: the entire state set consists of approximately $\mathcal{O}(10^{10})$ states, while the subset of states that is most relevant for our purpose, *i.e.*, the states that are reachable from an initial state corresponding to a particular growth state of the bacteria, still consists of $\mathcal{O}(10^4)$ states. It is obvious that Kripke structures of this size cannot be analyzed by visual inspection, and that formal verification techniques are needed to get a better insight into the transient and asymptotic

$$
\begin{array}{lcl}
\dot{x}_{crp} & = & \kappa^1_{crp} \\
& + & \kappa^2_{crp}\ s^-(x_{fis},\theta^2_{fis})\ s^+(x_{crp},\theta^1_{crp})\ s^+(x_{cya},\theta^1_{cya})\ s^+(u_{signal},\theta_{signal}) \\
& + & \kappa^3_{crp}\ s^-(x_{fis},\theta^1_{fis}) \\
& - & \gamma_{crp}\ x_{crp}
\end{array}
$$
$zero_{crp} < \theta^1_{crp} < \kappa^1_{crp}/\gamma_{crp} < (\kappa^1_{crp}+\kappa^2_{crp})/\gamma_{crp} < \theta^2_{crp} < \theta^3_{crp} < (\kappa^1_{crp}+\kappa^3_{crp})/\gamma_{crp} < (\kappa^1_{crp}+\kappa^2_{crp}+\kappa^3_{crp})/\gamma_{crp} < max_{crp}$

$$
\begin{array}{lcl}
\dot{x}_{cya} & = & \kappa^1_{cya} \\
& + & \kappa^2_{cya}\ (1 - s^+(x_{crp},\theta^3_{crp})\ s^+(x_{cya},\theta^3_{cya})\ s^+(u_{signal},\theta_{signal})) \\
& - & \gamma_{cya}\ x_{cya}
\end{array}
$$
$zero_{cya} < \theta^1_{cya} < \kappa^1_{cya}/\gamma_{cya} < \theta^2_{cya} < \theta^3_{cya} < (\kappa^1_{cya}+\kappa^2_{cya})/\gamma_{cya} < max_{cya}$

$$
\begin{array}{lcl}
\dot{x}_{fis} & = & \kappa^1_{fis}\ (1 - s^+(x_{crp},\theta^1_{crp})\ s^+(x_{cya},\theta^1_{cya})\ s^+(u_{signal},\theta_{signal}))\ s^-(x_{fis},\theta^5_{fis}) \\
& + & \kappa^2_{fis}\ s^+(x_{gyrAB},\theta^1_{gyrAB})\ s^-(x_{gyrI},\theta^2_{gyrI})\ s^-(x_{topA},\theta^2_{topA})\ s^-(x_{fis},\theta^5_{fis}) \\
& & \quad (1 - s^+(x_{crp},\theta^1_{crp})\ s^+(x_{cya},\theta^1_{cya})\ s^+(u_{signal},\theta_{signal})) \\
& - & \gamma_{fis}\ x_{fis}
\end{array}
$$
$zero_{fis} < \theta^1_{fis} < \kappa^1_{fis}/\gamma_{fis} < \theta^2_{fis} < \theta^3_{fis} < \theta^4_{fis} < \theta^5_{fis} < (\kappa^1_{fis}+\kappa^2_{fis})/\gamma_{fis} < max_{Fis}$

$$
\begin{array}{lcl}
\dot{x}_{gyrAB} & = & \kappa_{gyrAB}\ (1 - s^+(x_{gyrAB},\theta^2_{gyrAB})\ s^-(x_{gyrI},\theta^1_{gyrI})\ s^-(x_{topA},\theta^1_{topA})\ s^-(x_{fis},\theta^4_{fis})) \\
& - & \gamma_{gyrAB}\ x_{gyrAB}
\end{array}
$$
$zero_{gyrAB} < \theta^1_{gyrAB} < \theta^2_{gyrAB} < \kappa_{gyrAB}/\gamma_{gyrAB} < max_{gyrAB}$

$$
\begin{array}{lcl}
\dot{x}_{gyrI} & = & \kappa_{gyrI}\ s^+(x_{crp},\theta^1_{crp})\ s^+(x_{cya},\theta^1_{cya})\ s^+(u_{signal},\theta_{signal})\ s^+(x_{rpoS},\theta_{rpoS}) \\
& - & \gamma_{gyrI}\ x_{gyrI}
\end{array}
$$
$zero_{gyrI} < \theta^1_{gyrI} < \theta^2_{gyrI} < \kappa_{gyrI}/\gamma_{gyrI} < max_{gyrI}$

$$
\begin{array}{lcl}
\dot{x}_{rpoS} & = & \kappa_{rpoS} \\
& - & (\gamma^1_{rpoS} + \gamma^2_{rpoS}\ s^+(x_{rssB},\theta_{rssB})\ s^-(u_{signal},\theta_{signal}))\ x_{rpoS}
\end{array}
$$
$zero_{rpoS} < \kappa_{rpoS}/(\gamma^1_{rpoS}+\gamma^2_{rpoS}) < \theta_{rpoS} < \kappa_{rpoS}/\gamma^1_{rpoS} < max_{rpoS}$

$$
\begin{array}{lcl}
\dot{x}_{rssB} & = & \kappa^1_{rssB} \\
& + & \kappa^2_{rssB}\ s^+(x_{rpoS},\theta_{rpoS}) \\
& - & \gamma_{rssB}\ x_{rssB}
\end{array}
$$
$zero_{rssB} < \theta_{rssB} < \kappa^1_{rssB}/\gamma_{rssB} < (\kappa^1_{rssB}+\kappa^2_{rssB})/\gamma_{rssB} < max_{rssB}$

$$
\begin{array}{lcl}
\dot{u}_{signal} & = & 0
\end{array}
$$
$0 < \theta_{signal} < max_{signal}$

$$
\begin{array}{lcl}
\dot{x}_{topA} & = & \kappa^1_{topA}\ s^+(x_{gyrAB},\theta_{gyrAB})\ s^-(x_{gyrI},\theta^1_{gyrI})\ s^-(x_{topA},\theta^1_{topA})\ s^+(x_{fis},\theta^4_{fis}) \\
& + & \kappa^2_{topA}\ s^+(x_{rpoS},\theta_{rpoS}) \\
& - & \gamma_{topA}\ x_{topA}
\end{array}
$$
$zero_{topA} < \theta^1_{topA} < \theta^2_{topA} < \kappa^1_{topA}/\gamma_{topA} < \kappa^2_{topA}/\gamma_{topA} < (\kappa^1_{topA}+\kappa^2_{topA})/\gamma_{topA} < max_{topA}$

$$
\begin{array}{lcl}
\dot{x}_{rrn} & = & \kappa^2_{rrn} \\
& + & \kappa^1_{rrn}\ s^+(x_{fis},\theta^3_{fis}) \\
& - & \gamma_{rrn}\ x_{rrn}
\end{array}
$$
$zero_{rrn} < \kappa^2_{rrn}/\gamma_{rrn} < \theta_{rrn} < (\kappa^2_{rrn}+\kappa^1_{rrn})/\gamma_{rrn} < max_{rrn}$

Table 6.1: Piecewise-linear differential equations and parameters inequalities for the carbon starvation response network in *E. coli*.

dynamics of the network.

6.1.3 Analysis and verification of the carbon starvation response in *E. coli*

Several questions could be presented here in order to illustrate the verification of biological properties. However, only a few properties are presented, enough to illustrate each of the previously defined biological patterns (Chapter 4) and the CTRL language (Chapter 3). In particular, properties related to the conditions of reachability of each of the steady state attractors are explored.

6.1.3.1 Mutual inhibition of Fis and CRP

The regulatory protein CRP is the target of a signal-transduction pathway, which activates the adenylate cyclase Cya in case of carbon starvation. In turn, the latter synthesizes a small molecule, cAMP, which binds to CRP. This active form of CRP is able to regulate the expression of a large number of genes. In particular, CRP·cAMP binds to the promoter region of the gene *fis*, thereby preventing synthesis of new Fis proteins. Fis is an important regulator of genes involved in the cellular metabolism but it also inhibits *crp* expression, by binding to multiple sites in the two promoter regions of the gene, P1 and P2 (Figure 6.1). The regulatory interactions between genes *fis* and *crp* form a positive feedback loop, a motif often found in the genetic regulatory networks. When present in isolation, this kind of motif has been shown to lead to bistability [63]. In particular, it excludes simultaneous high or low levels of Fis and CRP at steady state. The question can be asked whether the motif is also functional in the context of the carbon starvation response network. For instance, the expression of *fis* is not only controlled by CRP·cAMP, but also by the DNA supercoiling level and Fis itself.

In order to answer this question, we first need to identify the elements that refer to the state of the biological system and stating these as atomic propositions. We start by creating an atomic proposition $high\_CRP$ to represent the accumulation of CRP to its maximum level where $x_{crp} \geq (\kappa^1_{crp} + \kappa^2_{crp} + \kappa^3_{crp})/\gamma_{crp}$. Then we create another atomic proposition low_CRP to restrict the concentrations values to a low level where $x_{crp} \leq \kappa^1_{crp}/\gamma_{crp}$. Analogously, we create an atomic proposition $high\_Fis$ to represent the accumulation of Fis to a high level where $x_{fis} \geq \kappa^4_{fis}/\gamma_{fis}$, and an atomic proposition low_Fis to restrict the concentration values of Fis to a low level where $x_{fis} \leq \kappa^1_{fis}/\gamma_{fis}$.

After the specification of these atomic propositions we then proceed with the formulation of the biological properties to be verified. We choose two *occurrence/exclusion* patterns to check both for the exclusion of states with high levels of CRP and Fis and states with low levels of CRP and Fis. The *occurrence/exclusion* patterns are then instantiated with the previously defined atomic propositions:

It	*is not possible*	for a state	$high\_CRP$ and $high\_Fis$	to occur

It	*is not possible*	for a state	low_CRP and low_Fis	to occur

The patterns are then automatically translated into the following CTL formulas:

$$\neg \mathsf{EF}(high\_CRP \wedge high\_Fis) \tag{6.1}$$

$$\neg \mathsf{EF}(low\_CRP \wedge low\_Fis) \tag{6.2}$$

The model checker NuSmv returns true in less than 2 seconds for both properties, confirming the mutual exclusion of Fis and CRP, meaning that the positive feedback loop involving *fis* and *crp* is functional.

One could now be interested in studying the conditions for the bistability of the system. In particular, the influence of the carbon starvation signal in the transitions between exponential and stationary phase. In order to verify this, we first need to characterize both attractors in GNA by means of atomic propositions. The stationary-phase attractor a_{stat} is characterized by a low basal expression level of the stable RNAs ($x_{rrn} \leq \theta_{rrn}$), an indicator of growth arrest. The exponential-phase attractor a_{exp} is characterized by a high expression level of the stable RNAs ($x_{rrn} > \theta_{rrn}$), characteristic of the high growth rate in exponential phase. To denote the presence of the carbon starvation input signal, the atomic proposition *sig* is characterized by low values of variable u_{signal} ($u_{signal} < \theta_{signal}$). Considering these atomic propositions, one can specify the following properties using the *consequence* pattern:

If a state	*sig*	occurs,	
then it is	*necessarily*	followed by a state	a_{stat}

If a state	$\neg sig$	occurs,	
then it is	*necessarily*	followed by a state	a_{exp}

The patterns are then automatically translated into the following CTL formulas:

$$\mathsf{AG}(sig \implies \mathsf{AF}(a_{stat})) \tag{6.3}$$

$$\mathsf{AG}(\neg sig \implies \mathsf{AF}(a_{exp})) \tag{6.4}$$

Choosing again the whole phase space for the initial conditions, the model checker returns true for both properties, indicating that for given nutrient conditions the attractors are mutually exclusive. More precisely, if *sig* is present (nutrient depletion) then the stationary-phase attractor is inevitably reached, whereas in the absence of *sig* (nutrient availability), the sys-

tem necessarily evolves towards the exponential-phase attractor. In mathematical terms, the system is therefore not bistable, but has a monostable steady-state response to each of the two possible inputs (nutrient depletion *vs.* nutrient availability).

6.1.3.2 Control of entry into stationary phase by RpoS

One of the extensions of this model relative to the one described by Ropers *et al.* [118], is the inclusion of the sigma factor RpoS. A question that naturally arises is to know what role RpoS plays in the dynamics of the system (This question was answered in Chapter 5 as a running example for the service-oriented architecture, but it is developed here in more detail).

RpoS or σ^s is a sigma factor that allows cells to adapt to and survive under harmful conditions by expressing a variety of stress response genes [70]. Due to its key role in the cell, the concentration of RpoS is tightly regulated at the transcriptional, translational, and post-translational levels. This model focuses on the conditions of stability of the protein. While cells grow on a carbon source, RpoS is actively degraded through the protein RssB, which binds to RpoS and targets the factor to an intracellular protease (Figure 6.1). However, the depletion of the carbon source inactivates RssB, thus allowing RpoS to accumulate to a high concentration. Given the important role of RpoS for the survival of the cell, one may ask whether the entry into stationary phase upon carbon starvation is always preceded by the accumulation of RpoS in the cell.

The first step in answering this question using the formal verification module of GNA consists in identifying elements of the question that refer to the state of the biological system and in stating these as atomic propositions. We represent the entry into stationary phase of the system by a low level of stable RNAs encoded by the *rrn* operons. This is motivated by the fact that stationary-phase cells do not need high levels of stable RNAs, contrary to what is required by the high translational activity in exponential phase. These characteristics are specified using the pattern-based property editor (Section 5.2.1), where we use the previously created atomic proposition a_{stat}, restricting the concentration values for the variable *rrn* to those below its (single) threshold θ_{rrn}. We also introduce an atomic proposition *high_RpoS*, representing the accumulation of RpoS to a value above its (single) threshold θ_{rpoS} (Figure B.1).

The second step is the formulation of the biological property using the pattern-based property editor and translator. We choose the *sequence* pattern to account for the temporal ordering of the two states: stationary phase and high expression of RpoS. The sequence pattern is instantiated by selecting the previously defined atomic propositions (Figure B.2):

A state	a_{stat}	is reachable and is		
necessarily	preceded	*at some time*	by a state	*high_RpoS*

Once the pattern is fully instantiated, it is automatically translated into the corresponding CTL formula:

$$\mathsf{EF}\ (a_{stat}) \wedge \neg \mathsf{E}\ (\neg high\_RpoS\ \mathsf{U}\ a_{stat})$$

After the specification of the property, one passes to the verification stage. For this step, the verification request must be configured in the verification window (Figure 5.4). First, we choose the name and version of the model checker plugin to be used (version 1.0 of the NUSMV implicit plugin was used in the example). Second, we specify the initial conditions. The resulting implicit FSTS represents the transition from exponential phase to stationary phase, starting from initial conditions corresponding to carbon depletion. We then run the verification request, which is treated by the service-oriented architecture as described in Chapter 5.

In response to the query the model checker returns false after 4 seconds. It means that the entry into stationary phase is not always preceded by the accumulation of RpoS in the cell. The counterexample is presented to the user as shown in the left panel of Figure 5.5. It consists of a subgraph of the initial FSTS, starting from the specified initial state and ending in a state where the property fails. By selecting a path in this subgraph, GNA allows the qualitative changes in the concentration of all the variables to be displayed (right panel of Figure 5.5). Looking at the evolution of the variables we immediately observe that there is (at least) one sequence of states leading to a low expression level of the *rrn* operons without having previously passed through a state with a high concentration of RpoS. This illustrates the negative verification result, and witnesses that the down-regulation of the stable RNAs does not require the previous accumulation of RpoS.

6.1.3.3 Expression of *topA* during growth-phase transitions

The model described by Ropers *et al.* [118] was not capable of accounting for the control of DNA supercoiling during growth-phase transition. In particular, TopA was predicted to be never expressed, which is inconsistent with published data [114]. The extension of the model with RpoS makes it possible to refine the description of the control of the DNA supercoiling level. On the one hand, GyrAB activity is regulated by GyrI, as mentioned previously, and on the other hand, the *topA* promoter is activated by RpoS. One question that arises is if *topA* is expressed in response to the carbon source availability?

In order to answer this question we verify its negation. In particular, we check if the absence of *topA* expression persists throughout the evolution of the system. First, we start by creating an atomic proposition *low_topA* representing the absence of *topA* expression where its value is $x_{topA} \leq \theta^1_{topA}$. We then proceed by choosing the *invariance* pattern to verify if *topA* expression maintains the value *low_topA*:

A state	*low_topA*	*can*	persist indefinitely

The pattern is then automatically translated into the following CTL formula:

$$\mathsf{EG}(low\_topA) \tag{6.5}$$

For the verification stage, we choose the name and version of the model checker plugin to be used (version 1.0 of the NUSMV implicit plugin was used in this example). Then, we specify the initial conditions corresponding to the exponential-phase conditions. The resulting implicit FSTS represents the transition from exponential phase to stationary phase. In response to the query the model checker returns false after 3 seconds, showing that the expression of *topA* is stimulated at the entry into stationary phase, most likely under the influence of RpoS. Indeed, following carbon starvation, the protein RssB is inactivated, which leads to the accumulation of RpoS at high levels. RpoS in turn stimulates the expression of *topA*.

6.1.3.4 Damped oscillations after carbon upshift

In the carbon starvation response network there is a negative feedback loop, involving the genes *gyrAB*, *topA*, and *fis* (Figure 6.1). GyrAB is a gyrase protein which supercoils the DNA structure, whereas the topoisomerase TopA relaxes it. An increase of the DNA supercoiling level stimulates expression of Fis, which in turn decreases the supercoiling level, by stimulating *topA* expression and inhibiting *gyrAB* expression. The resulting negative feedback loop was predicted to give rise to (damped) oscillations of Fis, GyrAB and stable RNA concentrations after a carbon upshift [118]. In the version of the carbon starvation model presented in this thesis, additional interactions contribute to controlling the DNA supercoiling level. Hence, the gyrase inhibitor GyrI represses the activity of GyrAB by forming a complex with the protein. The expression of *gyrI* is notably stimulated by RpoS [112]. An important validation of this model is to check whether the carbon upshift is still a necessary condition for the occurrence of damped oscillations.

To check for the occurrence of oscillations we can proceed in two distinct ways. The first way makes use of the GNA's state labeling procedure upon generation of the explicit FSTS, where each state of the FSTS that is identified as belonging to a (terminal) cycle is labeled with that information. A user can therefore define an atomic proposition *isOscillatoryState* making use of that particular information. By using the previously defined atomic proposition *sig* to indicate the presence of the carbon starvation signal, one can instantiate the *consequence* pattern as follows:

If a state	$\neg sig$	occurs,	
then it is	*necessarily*	followed by a state	$isOscillatoryState$

with the pattern being automatically translated into the following CTL formula:

$$\mathsf{AG}(\neg sig \implies \mathsf{AF}(isOscillatoryState)) \tag{6.6}$$

The second way to check for the occurrence of oscillations is through the use of the CTRL language (defined in Chapter 3). Since it is a more expressive language than CTL, one can characterize a specific pattern in the oscillation, like the (damped) oscillations of the stable RNAs concentration when entering exponential phase. We create the atomic proposition *dec_rrn* (*inc_rrn*) to represent a decreasing (increasing) concentration of the stable RNAs, and we specify the query with the following CTRL formula:

$$\mathsf{AG}_{\neg sig.\mathsf{true}^*}\mathsf{EF}^{\infty}_{(\mathsf{true}^*.inc\_rrn^+.\mathsf{true}^*.dec\_rrn^+)^+} \tag{6.7}$$

For the verification stage, we specify the initial conditions corresponding to the stationary-phase conditions. The resulting explicit FSTS represents the transition from stationary phase to exponential phase. We then choose the name and version of the model checker plugin to be used. The NUSMV explicit plugin for the CTL formula 6.6, and the CADP explicit plugin for the CTRL formula 6.7. In response to both queries the model checker returns true in 9 and 45 seconds respectively, showing that in this model extension damped oscillations still occur following a carbon upshift.

In Section 6.1.3.1 we concluded that the system was monostable with a mutual exclusion of Fis and CRP concentrations. One could use that information to confirm the conditions that drive the system towards the terminal cycle. Stationary phase (the initial condition) is characterized by a high concentration of CRP and a low concentration of Fis, and exponential phase (terminal cycle) is characterized by a low concentration of CRP and a high concentration of Fis. By creating the atomic proposition *dec_CRP* which looks at the sign of the derivative of the CRP variable, one can represent its decrease of concentration. The following CTRL formula checks for the existence of a path passing through a state where the concentration of CRP decreases, reaching exponential phase where the terminal cycle is characterized by the oscillation of the concentration of the stable RNAs:

$$\mathsf{AG}_{\neg sig.true^*.dec\_CRP^+.a_{exp}}\mathsf{EF}^{\infty}_{(\mathsf{true}^*.inc\_rrn^+.\mathsf{true}^*.dec\_rrn^+)^+} \tag{6.8}$$

Considering the same (stationary phase) initial conditions and the same model checker (CADP explicit) plugin, one can proceed with the verification of property 6.8. The CTRL model checker takes more than 2 minutes to return the verdict true, suggesting that the decrease

of the concentration of CRP drives the system towards the terminal cycle. Interestingly, the verification module inside GNA has the corresponding verification witness available. The user can visualize the path of states starting from the initial conditions at stationary phase leading to the terminal cycle. Also, by selecting these states the user has the possibility to observe the evolution of each variable of the system throughout this path, visualizing the decrease of the concentration of CRP and the oscillation of the concentration of the stable RNAs.

6.2 Analysis of the *FLR1* transcriptional response to mancozeb stress in *S. cerevisiae*

6.2.1 *FLR1* stress response to mancozeb in *S. cerevisiae*

Drugs, pesticides and other chemical compounds are not usually present in the natural ecosystems of most living organisms. However, multidrug resistance (MDR), *i.e.* the ability to acquire simultaneous resistance to unrelated chemical compounds, is a widespread phenomenon. This capability is often the result of the activation of multidrug efflux pumps which are able to catalyze the extrusion of chemically and structurally diverse substrates from the cytosol to the outer medium [79, 119]. Upon exposure to drugs and other chemical stresses, cells respond through the transcriptional up-regulation of genes encoding these MDR transporters. Long term multidrug resistance is acquired usually through the arising of point mutations leading to the constitutive activation of these proteins or to the constitutive activation or over-expression of their transcriptional activators [47, 131]. Understanding how multidrug efflux pumps are regulated at the transcriptional level is therefore crucial to define suitable strategies to overcome drug resistance, including the definition of new drug targets and the design of new drugs.

Transcriptional regulation has been extensively studied in the model eukaryotic organism *S. cerevisiae*. Although much knowledge has been gathered on the effect of individual transcription factors in individual target genes, little is still known on how transcription factors interact and cross-talk to produce the registered transcriptional responses. The few examples studied in more detail [96, 97, 129] strongly suggest that, in most cases, transcriptional control results from the balance of the action of diverse transcription factors which may act alone or as interacting partners, may regulate and be regulated by other transcription factors and may control their own expression through complex feedback loops [93].

Figure 6.2 presents a specific network of five (including one unidentified) transcription factors regulating the transcriptional activation of the *FLR1* gene in yeast cells exposed to the agricultural fungicide mancozeb[2]. *FLR1* encodes a plasma membrane member of the

[2]In this subsection, the notation used for the gene and protein names are those consistent with the community working with the *S. cerevisiae*: use of italic majuscules for gene names (*i.e.*, *PDR3* and *RPN4*) and non-italic for protein names (*i.e.*, Pdr3 and Rpn4).

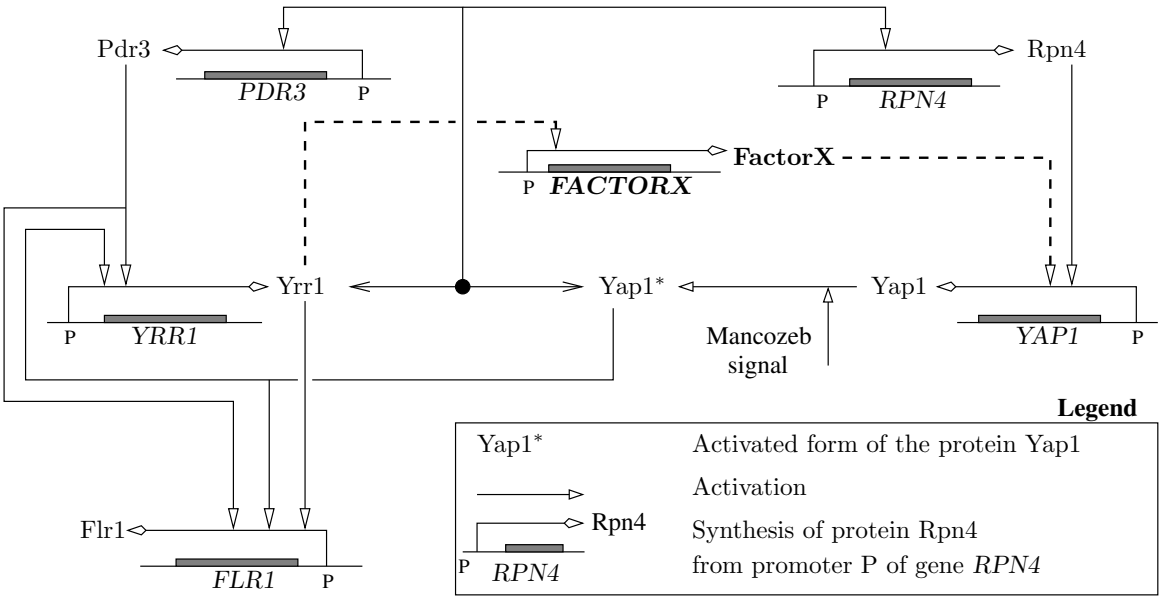


Figure 6.2: Network of key genes, proteins and regulatory interactions involved in the *FLR1* response in the presence of mancozeb in *S. cerevisiae* [106].

12-spanner Drug:H^+ Antiporter family 1 of the Major Facilitator Superfamily [119], being involved in the multidrug resistance phenomenon by conferring resistance to drugs and other inhibitory chemicals [2, 23, 111, 129, 130]. *FLR1* transcriptional activation was registered upon exposure to drugs, oxidants and pro-oxidants, being fully dependent on the bZIP transcription factor Yap1 [2, 111, 130], implicated in oxidative stress response and pleiotropic drug resistance. However, maximal levels of *FLR1* induction by the agricultural fungicide benomyl were only observed in the presence of Pdr3, but where independent of Pdr1 [23, 130]. Pdr1 and Pdr3 being the first described major regulators of the MDR phenomenon in yeast. A fourth transcription factor, Rpn4, controlling the expression of proteosome genes, was seen to be further required for maximal *FLR1* up-regulation in yeast cells challenged with mancozeb [129]. The importance of these transcription factors in yeast resistance and response to mancozeb was stressed in previous chemogenomics [48] and expression proteomics [121] analysis.

6.2.2 Model of *FLR1* stress response to mancozeb in *S. cerevisiae*

Similarly to the *E. coli* network, it is not possible to build a quantitative model of the *S. cerevisiae* network, due to the lack of biological data. We used the same approach to build a qualitative model, by specifying the rules of the dynamics of the elements of the system. The qualitative dynamics of the network are described by six coupled piecewise-linear differential equations and forty eight inequality constraints on the parameter values (Table 6.2) [106]. The

qualitative analysis of this network generates Kripke structures of the order of approximately $\mathcal{O}(10^5)$ states, while the subset of states that is most relevant for our purpose, *i.e.*, the states that are reachable from an initial state corresponding to the low concentration of the *FLR1* gene, still consists on $\mathcal{O}(10^2)$ states. Even though the qualitative behaviors of this network are significantly smaller than the previous network, it is still difficult to analyze them by visual inspection, and formal verification techniques are still required to get a better insight into the transient and asymptotic dynamics of the network.

6.2.3 Analysis and verification of the *FLR1* stress response to mancozeb in *S. cerevisiae*

In this section, generic properties concerning the reachability of the attractors of the system are presented, as well as properties concerning the validation of particular interactions.

6.2.3.1 Attractors reachability

We start by first looking into the existence of attractors in the system, and the conditions for their reachability. Using the attractor search functionality [46] available in the GNA modeling and simulation tool, two attractors were identified. The first attractor identified by means of GNA is a state in the Kripke structure corresponding to an asymptotically stable steady state of the piecewise-linear model. This state is characterized by a low expression of all the transcription factors and the *FLR1* gene. This is due to the fact that the signal corresponding to the entry of mancozeb into the cytosol is absent. The second attractor corresponds to another steady state of the piecewise-linear model. This state is characterized by a presence of the mancozeb signal and a high expression of all the transcription factors and of the *FLR1* gene. This is consistent with the fact that the presence of the mancozeb signal will activate the cascade of regulators that will increase the expression of *FLR1*.

In order to verify the conditions for the reachability of these attractors, we start by characterizing them in GNA by means of atomic propositions. The steady state corresponding to a low expression of *FLR1* ($x_{flr1} \leq \kappa^b_{flr1}$) is labeled a_{normal}. The steady state corresponding to a maximal expression of *FLR1* ($x_{flr1} \geq (\kappa^b_{flr1} + \kappa^1_{flr1} + \kappa^2_{flr1} + \kappa^3_{flr1})/\gamma_{flr1}$) is labeled $a_{response}$. The *occurrence* pattern is then picked to check for the reachability of both steady states, and we instantiate them with the corresponding atomic propositions (Figure B.3):

It	*is possible*	for a state	a_{normal}	to occur

It	*is possible*	for a state	$a_{response}$	to occur

$\dot{u}_{mancozeb}$	=	0
$0 < \theta_{mancozeb} < max_{mancozeb}$		
$\dot{x}_{pdr3}$	=	κ^b_{pdr3}
	+	$\kappa^1_{pdr3}\ s^+(x_{yrr1}, \theta^1_{yrr1})\ s^+(x_{yap1}, \theta^1_{yap1})\ s^+(u_{mancozeb}, \theta_{mancozeb})$
	-	$\gamma_{pdr3}\ x_{pdr3}$
$0 < \kappa^b_{pdr3}/\gamma_{pdr3} < \theta^1_{pdr3} < (\kappa^b_{pdr3} + \kappa^1_{pdr3})/\gamma_{pdr3} < max_{pdr3}$		
$\dot{x}_{rpn4}$	=	κ^b_{rpn4}
	+	$\kappa^1_{rpn4}\ s^+(x_{yrr1}, \theta^1_{yrr1})\ s^+(x_{yap1}, \theta^1_{yap1})\ s^+(u_{mancozeb}, \theta_{mancozeb})$
	-	$\gamma_{rpn4}\ x_{rpn4}$
$0 < \theta^1_{rpn4} < \kappa^b_{rpn4}/\gamma_{rpn4} < \theta^2_{rpn4} < (\kappa^b_{rpn4} + \kappa^1_{rpn4})/\gamma_{rpn4} < max_{rpn4}$		
$\dot{x}_{yap1}$	=	κ^b_{yap1}
	+	$\kappa^1_{yap1}\ s^+(u_{mancozeb}, \theta_{mancozeb})$
	+	$\kappa^2_{yap1}\ s^+(x_{rpn4}, \theta^1_{rpn4})$
	+	$\kappa^3_{yap1}\ \boldsymbol{s^+(x_{factorX}, \theta_{factorX})}$
	-	$\gamma_{yap1}\ x_{yap1}$
$0 < \theta^b_{yap1} < \kappa^b_{yap1}/\gamma_{yap1} < \theta^1_{yap1} < (\kappa^b_{yap1} + \kappa^1_{yap1})/\gamma_{yap1} < (\kappa^b_{yap1} + \kappa^2_{yap1})/\gamma_{yap1} < (\kappa^b_{yap1} + \kappa^1_{yap1} + \kappa^2_{yap1})/\gamma_{yap1} < \theta^2_{yap1} < (\kappa^b_{yap1} + \kappa^3_{yap1})/\gamma_{yap1} < (\kappa^b_{yap1} + \kappa^1_{yap1} + \kappa^3_{yap1})/\gamma_{yap1} < (\kappa^b_{yap1} + \kappa^2_{yap1} + \kappa^3_{yap1})/\gamma_{yap1} < \theta^3_{yap1} < (\kappa^b_{yap1} + \kappa^1_{yap1} + \kappa^2_{yap1} + \kappa^3_{yap1})/\gamma_{yap1} < max_{yap1}$		
$\dot{x}_{yrr1}$	=	κ^b_{yrr1}
	+	$\kappa^1_{yrr1}\ s^+(x_{yap1}, \theta^b_{yap1})\ s^+(u_{mancozeb}, \theta_{mancozeb})$
	+	$\kappa^2_{yrr1}\ s^+(x_{pdr3}, \theta^1_{pdr3})$
	+	$\kappa^3_{yrr1}\ s^+(x_{yap1}, \theta^3_{yap1})\ s^+(u_{mancozeb}, \theta_{mancozeb})$
	-	$\gamma_{yrr1}\ x_{yrr1}$
$0 < \kappa^b_{yrr1}/\gamma_{yrr1} < \theta^1_{yrr1} < (\kappa^b_{yrr1} + \kappa^1_{yrr1})/\gamma_{yrr1} < \theta^2_{yrr1} < (\kappa^b_{yrr1} + \kappa^2_{yrr1})/\gamma_{yrr1} < (\kappa^b_{yrr1} + \kappa^1_{yrr1} + \kappa^2_{yrr1})/\gamma_{yrr1} < \theta^3_{yrr1} < (\kappa^b_{yrr1} + \kappa^3_{yrr1})/\gamma_{yrr1} < (\kappa^b_{yrr1} + \kappa^1_{yrr1} + \kappa^3_{yrr1})/\gamma_{yrr1} < (\kappa^b_{yrr1} + \kappa^2_{yrr1} + \kappa^3_{yrr1})/\gamma_{yrr1} < (\kappa^b_{yrr1} + \kappa^1_{yrr1} + \kappa^2_{yrr1} + \kappa^3_{yrr1})/\gamma_{yrr1} < max_{yrr1}$		
$\boldsymbol{\dot{x}_{factorX}}$	=	$\boldsymbol{\kappa^b_{factorX}}$
	+	$\boldsymbol{\kappa^1_{factorX}\ s^+(x_{yrr1}, \theta^1_{yrr1})\ s^+(u_{mancozeb}, \theta_{mancozeb})}$
	-	$\boldsymbol{\gamma_{factorX}\ x_{factorX}}$
$\boldsymbol{0 < \kappa^b_{factorX}/\gamma_{factorX} < \theta_{factorX} < (\kappa^b_{factorX} + \kappa^1_{factorX})/\gamma_{factorX} < max_{factorX}}$		
$\dot{x}_{flr1}$	=	κ^b_{flr1}
	+	$\kappa^1_{flr1}\ s^+(x_{yap1}, \theta^b_{yap1})\ s^+(u_{mancozeb}, \theta_{mancozeb})$
	+	$\kappa^2_{flr1}\ s^+(x_{pdr3}, \theta^1_{pdr3})\ s^+(x_{yap1}, \theta^1_{yap1})\ s^+(u_{mancozeb}, \theta_{mancozeb})$
	+	$\kappa^3_{flr1}\ s^+(x_{yrr1}, \theta^2_{yrr1})\ s^+(x_{yap1}, \theta^3_{yap1})\ s^+(u_{mancozeb}, \theta_{mancozeb})$
	-	$\gamma_{flr1}\ x_{flr1}$
$0 < \kappa^b_{flr1}/\gamma_{flr1} < (\kappa^b_{flr1} + \kappa^1_{flr1})/\gamma_{flr1} < (\kappa^b_{flr1} + \kappa^2_{flr1})/\gamma_{flr1} < (\kappa^b_{flr1} + \kappa^1_{flr1} + \kappa^2_{flr1})/\gamma_{flr1} < (\kappa^b_{flr1} + \kappa^3_{flr1})/\gamma_{flr1} < (\kappa^b_{flr1} + \kappa^1_{flr1} + \kappa^3_{flr1})/\gamma_{flr1} < (\kappa^b_{flr1} + \kappa^2_{flr1} + \kappa^3_{flr1})/\gamma_{flr1} < (\kappa^b_{flr1} + \kappa^1_{flr1} + \kappa^2_{flr1} + \kappa^3_{flr1})/\gamma_{flr1} < max_{flr1}$		

Table 6.2: Piecewise-linear differential equations and parameters inequalities for the *FLR1* stress response to mancozeb network in *S. cerevisiae*.

The patterns are then automatically translated into the following CTL formulas:

$$\mathsf{EF}(a_{normal}) \tag{6.9}$$

$$\mathsf{EF}(a_{response}) \tag{6.10}$$

For the verification stage, we choose the name and version of the model checker plugin to be used (version 1.0 of the NuSmv implicit plugin was used with these properties). Then we choose the whole phase space for the initial conditions. We then run the verification request, which is treated by the service-oriented architecture. The model checker returns true for both properties confirming the reachability of these two steady states in the model. We are now interested in verifying if their occurrence is dependent of the presence of the mancozeb signal. We introduce the atomic proposition sig_m to denote the presence of mancozeb. Both properties are specified using the *consequence* pattern (Figure B.4):

If a state	$\neg sig_m$	occurs,	
then it is	*necessarily*	followed by a state	a_{normal}

If a state	sig_m	occurs,	
then it is	*necessarily*	followed by a state	$a_{response}$

The patterns are then automatically translated into the following CTL formulas:

$$\mathsf{AG}(\neg sig_m \implies \mathsf{AF}(a_{normal})) \tag{6.11}$$

$$\mathsf{AG}(sig_m \implies \mathsf{AF}(a_{response})) \tag{6.12}$$

Choosing again all the phase space for the initial conditions, the model checker returns true for both properties, indicating that the attractors are mutually exclusive and dependent on the presence of the mancozeb signal. Like the previous system, in mathematical terms the *FLR1* response is not bistable, but has a monostable response to each of the possible values for the input variable (mancozeb presence *vs.* mancozeb absence).

6.2.3.2 Pdr3 necessity for *FLR1* maximum expression

Like previously mentioned, the full activation of *FLR1* is dependent on Yap1, Yrr1, Pdr3 and Rpn4 [129]. The effect of Yap1 and Yrr1 in *FLR1* transcription is known to be direct [42, 97]. Rpn4-binding locus cannot be found in the *FLR1* promoter region, so Rpn4 is assumed to influence *FLR1* through Yap1, whose expression is dependent on Rpn4 in presence of mancozeb [129]. Although the Pdr3-binding locus can be found in the *FLR1* promoter region, there is still no experimental evidence to support this hypothesis. We made the assumption

that Pdr3 acts as co-transcription factor of Yrr1 when regulating *FLR1*, and we are interested in confirming that the model accounts for the fact that the full activation of *FLR1* is dependent on Pdr3.

In order to answer this question, we first state the biological elements as atomic propositions in GNA. An atomic proposition $a_{response}$ was already defined to represent the maximal expression of *FLR1*. We introduce a new atomic proposition $high\_Pdr3$ to represent a high expression of *PDR3*, restricting the concentration values for the variable $pdr3$ to those above its threshold $x_{pdr3} \geq \theta_{pdr3}$.

We choose the *sequence* pattern to account for the temporal ordering of these two events: maximal expression of *FLR1* and high expression of *PDR3*. The sequence pattern is then instantiated with these atomic propositions:

A state	$a_{response}$	is reachable and			
is	*necessarily*	preceded	*at some time*	by a state	$high\_Pdr3$

Once the pattern is fully instantiated, it is automatically translated into the corresponding CTL formula:

$$\mathsf{EF}\ (a_{response}) \wedge \neg \mathsf{E}\ (\neg high\_Pdr3\ \mathsf{U}\ a_{response})$$

We choose the NUSMV implicit plugin in the configuration window and we select a_{normal} steady state for the initial conditions, turning the sig_m on to represent the presence of mancozeb. In response to this query, the model checker returns true confirming that the maximal expression of *FLR1* is dependent on its co-regulation by Pdr3 and Yrr1 transcription factors.

6.2.3.3 Validation of a new Yap1 regulator

The verification of the previous property, suggested the acquisition of more biological data. In particular, double mutants knockout experiments in order to validate the interactions between the considered transcription factors.

The double deletion mutant $\Delta yrr1\Delta pdr3$ exhibits $yap1$ transcript levels coincident with those found in the $\Delta yrr1$ single deletion mutant, suggesting that Pdr3 influence on the *YAP1* up-regulation is dependent on Yrr1. Regarding the Rpn4 influence on *YAP1* up-regulation, the double deletion mutant $\Delta yrr1\Delta rpn4$ exhibits *YAP1* transcript levels qualitatively lower than those observed for the single mutants $\Delta yrr1$ and $\Delta rpn4$, meaning that there is an addictive effect of Yrr1 and Rpn4 in *YAP1* up-regulation. However, in the initial considered version of the model we had the assumption that Yrr1 could only influence *YAP1* up-regulation through Rpn4. This new information led us to revise a previous version of the model, including a new transcription factor FactorX (bold in Figure 6.2 and Table 6.2). Given that there is also no previous evidence for a direct role of Yrr1 in *YAP1* transcription and that there is

no Yrr1 binding site in the *YAP1* promoter region, an unknown transcription factor FactorX is proposed to perform this interaction between Yrr1 and *YAP1*.

In order to verify if the current model accounts for the new biological data, we simulate a $\Delta rpn4$ knockout in the model by restricting the values to zero of the equation corresponding to Rpn4. Then we create an atomic proposition low_Yap1 to restrict the concentration values of *YAP1* to a level where $x_{fis} < (\kappa^b_{yap1} + \kappa^3_{yap1})/\gamma_{yap1}$. We then proceed by choosing the *invariance* pattern to verify if *YAP1* expression is maintained at low levels through the mancozeb response.

A state	low_yap1	*must*	persist indefinitely

The pattern is then automatically translated into the following CTL formula:

$$\mathsf{AG}(low\_yap1) \tag{6.13}$$

We choose the version 1.0 of the NUSMV implicit plugin and the initial conditions corresponding to the steady state a_{normal}, turning the signal sig_m on to represent the presence of mancozeb. In response to this query, the model checker returns false together with a counterexample showing increase of *YRR1* expression, followed by an increase of $factorX$ expression, with the final increase of expression of *YAP1*. This means that the extension of the model with a new regulator of *YAP1* is sufficient to produce the behavior consistent with the biological data. In particular, the fact that a $\Delta rpn4$ knockout should not be sufficient to prevent the observation of some *YAP1* expression.

Once again, this property shows the necessity of further biological experiments in order to validate the assumption of the existence of a FactorX to control the expression of *YAP1*. However, the identification of this transcription factor remains to be established.

6.3 Conclusions

In this chapter two genetic regulatory networks from two model organisms were analyzed. Both networks have a model containing only a few number of variables, but they are enough to generate behaviors composed by thousands of states. It is clear that as the complexity and size of the models increase and more biological data becomes available, it becomes impractical to manually analyze the simulated behaviors. The use of automated methods like formal verification techniques based on model-checking becomes imperative for the automated validation of these behaviors.

The obtained results illustrate that the use of the proposed framework lowers the obstacles of using formal verification technology in systems biology. In particular, the natural usage of the developed set of patterns, which are sufficient to express most of the biological prop-

erties. Also, the CTRL language proves to be adequate whenever there is a necessity for a more expressive temporal logic to describe events occurring inside an oscillation, or to easily specify sequences of events through the use of regular expressions. The specification of these properties is not possible using only the CTL language.

Another important result is the better performance of verification when using an implicit representation with respect to using an explicit representation for the FSTSs containing all the system's behaviors. Properties verified using implicit representations never took more than 2-4 seconds, for the studied examples. This is due to the fact that implicit representations encode a transition function in order to delay the exploration of the states. This delay avoids the exhaustive enumeration of all the states in the phase space occurring in the simulation side, performing the exploration of only the necessary ones on the model checking side. Also, it avoids sending the resulting explicit FSTS through the web with the developed architecture. For example, the *E. coli* carbon starvation model generates an explicit FSTS bigger than 500Mb when considering all the phase space as initial conditions.

Finally, the use of these methods prove to be important, not only for the automated comparison of the simulated behaviors with current available biological data, but also for quickly testing new predictions before verifying them experimentally. New biological meaningful insights were obtained for both tested examples. For example, in the *E. coli* model we were able to indicate that the down-regulation of the stable RNAs does not require the previous accumulation of RpoS. Also, for the *S. cerevisiae* model we were able to indicate an unknown transcription factor that helps explaining the current available biological data.

Chapter 7

Conclusions

This chapter summarizes the achievements of my thesis and it presents some directions for future work.

7.1 Summary of achievements

The study of biological regulatory networks has led to the development of increasingly large and complex models. However, as the size of these models increases, it becomes infeasible to manually verify the predictions against experimental biological data or identify interesting features in dozens of simulation traces. This calls for the use of automated and scalable methods that help the modeler with the identification and verification of interesting dynamical properties of the network. Formal verification methods have historically been used for the verification of hardware and software systems. In recent years, several applications of formal verification methods to biological models have shown that these methods are a promising way to deal with the analysis of large and complex biological models.

Despite of the recent success of the application of formal verification techniques for the analysis of large biological models, it became clear that several problems and limitations exist, preventing these techniques from being naturally used by the systems biology community. In this PhD thesis I have three major achievements for alleviating at different levels the usage of model checking techniques for the automated verification of biological regulatory networks.

The first achievement is the definition of a new temporal logic adapted to the specification of biological concepts, the CTRL language. CTRL achieves a higher expressive power by extending CTL with regular expressions and fairness operators. It has the capability of expressing biological concepts like the multistability of several biological attractors or the identification and characterization of oscillations, which typically need to be specified with either branching-time or linear-time logics, respectively. It proves to be a suitable alternative to other temporal logics with high expressive power, like the propositional μ-calculus [84], obtaining a good compromise between expressive power, user-friendliness and complexity of

model checking. In addition, it is capable of specifying sequences of events (experimental observations) in a concise manner by using the power of regular expressions. Moreover, we were able to obtain a real on-the-fly model checker for CTRL by defining and implementing a translation from CTRL to HMLR, and by reusing the verification and diagnostic generation features of the EVALUATOR 3.6 model checker of CADP.

The formulation of properties in temporal logic is far from obvious, especially for non-expert users. In response to this problem, the formal verification community proposed the use of patterns [50], that is, high-level query templates that capture recurring questions in a specific application domain and that can be automatically translated to temporal logic. Inspired by this work, I proposed in a set of biological patterns formulated in structured natural language, to facilitate the use of formal verification methods in the systems biology field. These patterns capture a large number of frequently-asked questions by modelers in systems biology, without being restricted to a particular type of network, a particular biological system or a particular modeling tool. They were made available through a stand-alone Java application and an application programming interface (API) that can be integrated into other modeling tools, providing translations into three different temporal logics: CTL, CTRL and μ-calculus. Complex biological properties, like multistability or oscillations, still need to be specified using a temporal logic with high expressive power, like CTRL. However, this is consistent with the main idea underlying the use of patterns, namely that they cannot be expected to cover all possible queries, but rather should allow users to formulate frequent questions without worrying about their translation to temporal logic.

The third achievement consists of a service-oriented architecture for the integration of modeling tools with formal verification tools, providing a single integrated working environment. The proposed architecture is completely transparent for the modeler and it is platform-independent. Furthermore, its plugin system provides a modular way to add new formal verification methods without having to change the modeling tool. This architecture has been implemented in the context of GNA modeling tool, and with the NUSMV and CADP model checkers. GNA was extended with a verification module which connects the modeling tool to the architecture, also integrating the pattern-based property editor and providing graphical user interfaces to present the verification results to the modeler in an accessible way.

The applicability and validation of the developed methods was illustrated by the analysis of genetic regulatory networks of two model organisms, *E. coli* and *S. cerevisiae*. These two networks are too complex to allow manual verification of dynamic properties, which has motivated the application of the above-mentioned integrated modeling and verification environment. New insights were gained for both analyzed biological systems, like the extension of the *S. cerevisiae* model with a new transcription factor. This new information will now need to be confirmed by the acquisition of new biological data.

7.2 Directions of future work

As future work for this PhD thesis, one might think on some directions to improve and extend the proposed methods and their applications.

One immediate improvement regarding the representation of the model with FSTSs, is the specification of an implicit version to be used with the EVALUATOR model checker. This would enable to fully use the expressive power of the CTRL language for the specification of complex biological properties, without having to explicitly generate the FSTS. Additionally, this model representation would mean better verification performance, since the implicit representation of the model would be smaller and faster to transmit through the implemented architecture than its explicit counterpart.

There is also room for improvement regarding the usage of the information returned by the model checker. Current model checkers give a true/false verdict together with the corresponding witness/counterexample. This witness/counterexample is a path in the FSTS containing valuable information on the conditions for which the considered property was evaluated true or false. One might envisage the usage of this specific information for semi-automatic model revision. As model revision, we can consider the validation or refutation of some of the existing interactions in the model, or a specific ordering of the model parameters.

Another possible direction is the use of symbolic approaches for model encoding. During the modeling process, one may start with an incomplete model, since it is not always possible to have enough biological data to specify the complete ordering of the equation parameters. The symbolic representation of the model has the advantage of enabling the specification of only a partial order for these parameters. Such a model representation is in fact describing a general class of all the fully-specified sub-models. Symbolic model checkers can interpret these representations, verifying in this way the equivalent of a large number of qualitative models without explicitly enumerating them. One can also use this method to navigate through the classes of models by using the verdict returned by the model checker to discard or select individual models or sets of models.

Further tests on the usability of the developed interfaces are also necessary in order to assess the major needs of a larger audience of users. Most of the tests on usability were performed by either people closely related to the development of the GNA modeling and simulation tool, or by students without any previous contact with modeling and simulation tools or even modeling itself. There are however expert modeling users coming from other modeling and simulation tools that could provide some important insights, in particular, on the presentation and interpretation of verification results.

Finally, the scalability of the proposed methods should be tested by their application to the analysis of larger and more complex biological systems. Also, in addition to genetic regulatory networks, these methods should be tested with networks integrating different kinds of biological data (*e.g.*, signaling pathways or metabolic data). Such an example, is the work by

Baldazzi *et al.* [7] where the methods presented in this thesis were applied for the construction of a large model. The authors extended the previously described carbon starvation response model in *E. coli* integrating genetic and metabolic information.

Part IV

Appendices

Appendix A

Proofs of the translation from CTRL to MESS

A.1 Translation from CTRL to RESs

A few additional definitions and lemmas are required in order to prove Proposition 1. Given a propositional environment $\delta = [U_1/X_1, ..., U_n/X_n]$, its *support* is defined as $supp(\delta) = \{X_1, ..., X_n\}$, i.e., the set of variables that are mapped by δ to state sets. It is straightforward to show that, for environments with disjoint supports, the $\oslash$ operator is associative, commutative, and has the empty environment $[\,]$ as neutral element. Moreover, $supp([\![B]\!]_K\delta) = bv(B)$ and $supp([\![BL]\!]_K\delta) = bv(BL)$ for any Kripke structure K, equation block B, equation block list BL, and environment δ.

Lemma 2 *Let K be a Kripke structure, B an equation block, and δ_1, δ_2 two propositional environments such that $supp(\delta_1) \cap supp(\delta_2) = \emptyset$ and $fv(B) \subseteq supp(\delta_1)$. Then:*

$$[\![B]\!]_K(\delta_1 \oslash \delta_2) = [\![B]\!]_K\delta_1.$$

Proof Let $B = \{X_i \stackrel{\sigma}{=} \varphi_i\}_{1\leq i\leq n}$ be an equation block and δ_1, δ_2 two propositional environments as stated in the hypothesis. The semantics of B in the context of an environment δ is determined by the associated functional $\Phi_\delta : (2^S)^n \to (2^S)^n$ defined as follows:

$$\Phi_\delta(U_1, ..., U_n) = \langle [\![\varphi_i]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])\rangle_{1\leq i\leq n}$$

To prove the lemma, we show that the two functionals $\Phi_{(\delta_1\oslash\delta_2)}$ and Φ_{δ_1} are identical, i.e., $[\![\varphi]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n]) = [\![\varphi]\!]_K(\delta_1 \oslash [U_1/X_1, ..., U_n/X_n])$ for any formula φ and any $U_1, ..., U_n \subseteq S$. We proceed by structural induction on φ.

- $\varphi ::= p$:

$[\![p]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])$
$= \{s \in S \mid p \in L(s)\}$ by def. of $[\![\ ]\!]$
$= [\![p]\!]_K(\delta_1 \oslash [U_1/X_1, ..., U_n/X_n])$ by def. of $[\![\ ]\!]$.

- $\varphi ::= X$:

  Two cases are possible.

  1. $X \in \{X_1, ..., X_n\}$, i.e., X is bound in B. Let $i \in [1, n]$ such that $X = X_i$.
     $[\![X_i]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])$
     $= ((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])(X_i)$ by def. of $[\![\ ]\!]$
     $= [U_1/X_1, ..., U_n/X_n](X_i)$ by def. of $\oslash$
     $= U_i$ by def. of $[\ ]$
     $= [\![X_i]\!]_K(\delta_1 \oslash [U_1/X_1, ..., U_n/X_n])$ by def. of $[\![\ ]\!]$.
  2. $X \notin \{X_1, ..., X_n\}$, i.e., X is free in B. This means $X \in supp(\delta_1)$.
     $[\![X]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])$
     $= ((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])(X)$ by def. of $[\![\ ]\!]$
     $= (\delta_1 \oslash \delta_2)(X)$ by def. of $\oslash$
     $= \delta_1(X)$ $fv(B) \not\subseteq supp(\delta_2)$
     $= [\![X]\!]_K\delta_1$ by def. of $[\![\ ]\!]$
     $= [\![X]\!]_K(\delta_1 \oslash [U_1/X_1, ..., U_n/X_n])$ by def. of $\oslash$.

- $\varphi ::= \varphi_1 \vee \varphi_2$ (similarly for $\varphi ::= \varphi_1 \wedge \varphi_2$):

  $[\![\varphi_1 \vee \varphi_2]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])$
  $= ([\![\varphi_1]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])) \cup$
  $([\![\varphi_2]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n]))$ by def. of $[\![\ ]\!]$
  $= [\![\varphi_1]\!]_K(\delta_1 \oslash [U_1/X_1, ..., U_n/X_n]) \cup [\![\varphi_2]\!]_K((\delta_1 \oslash [U_1/X_1, ..., U_n/X_n])$
  by inductive hypothesis
  $= [\![\varphi_1 \vee \varphi_2]\!]_K(\delta_1 \oslash [U_1/X_1, ..., U_n/X_n])$ by def. of $[\![\ ]\!]$.

- $\varphi ::= \mathsf{EF}_\rho\varphi$ (similarly for $\varphi ::= \mathsf{AF}_\rho\varphi \mid \mathsf{EG}_\rho\varphi \mid \mathsf{AG}_\rho\varphi$):

  $[\![\mathsf{EF}_\rho\varphi]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])$
  $= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\pi_{0,i} \models_K \rho \wedge$
  $\pi_i \in [\![\varphi]\!]_K((\delta_1 \oslash \delta_2) \oslash [U_1/X_1, ..., U_n/X_n])\}$ by def. of $[\![\ ]\!]$
  $= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\pi_{0,i} \models_K \rho \wedge$
  $\pi_i \in [\![\varphi]\!]_K(\delta_1 \oslash [U_1/X_1, ..., U_n/X_n])\}$ by inductive hypothesis
  $= [\![\mathsf{EF}_\rho\varphi]\!]_K(\delta_1 \oslash [U_1/X_1, ..., U_n/X_n])$ by def. of $[\![\ ]\!]$.

- $\varphi ::= \mathsf{EF}^\infty_\rho$ (similarly for $\varphi ::= \mathsf{AF}^\infty_\rho \mid \mathsf{EG}^\dashv_\rho \mid \mathsf{AG}^\dashv_\rho$):

  $[\![\mathsf{EF}^\infty_\rho]\!]_K(\delta_1 \oslash \delta_2) = [\![\mathsf{EF}^\infty_\rho]\!]_K\delta_1$ because EF^∞_ρ is closed, so its interpretation is independent of any environment δ.

□

Lemma 3 *Let K be a Kripke structure and BL_1, BL_2 be two closed equation block lists. Then:*

$$[\![BL_1; BL_2]\!]_K = [\![BL_1]\!]_K \oslash [\![BL_2]\!]_K.$$

Proof Let K, BL_1, BL_2 as stated in the hypothesis. We proceed by structural induction on BL_1.

- $BL_1 ::= \varepsilon$:

$$\begin{aligned}
[\![\varepsilon; BL_2]\!]_K &= [\![BL_2]\!]_K && \text{by def. of ;} \\
&= [\,] \oslash [\![BL_2]\!]_K \\
&= [\![\varepsilon]\!]_K \oslash [\![BL_2]\!]_K && \text{by def. of } [\![\ ]\!].
\end{aligned}$$

- $BL_1 ::= B.BL_1$:

$$\begin{aligned}
[\![(B.BL_1); BL_2]\!]_K &= [\![B.(BL_1; BL_2)]\!]_K \\
&\qquad \text{by def. of ;} \\
&= [\![B]\!]_K([\![BL_1; BL_2]\!]_K) \oslash [\![BL_1; BL_2]\!]_K \\
&\qquad \text{by def. of } [\![\ ]\!] \\
&= [\![B]\!]_K([\![BL_1]\!]_K \oslash [\![BL_2]\!]_K) \oslash ([\![BL_1]\!]_K \oslash [\![BL_2]\!]_K) \\
&\qquad \text{by ind. hyp.} \\
&= [\![B]\!]_K([\![BL_1]\!]_K) \oslash ([\![BL_1]\!]_K \oslash [\![BL_2]\!]_K) \\
&\qquad \text{by Lemma 2} \\
&= ([\![B]\!]_K([\![BL_1]\!]_K) \oslash [\![BL_1]\!]_K) \oslash [\![BL_2]\!]_K \\
&\qquad \text{by assoc.} \\
&= [\![B.BL_1]\!]_K \oslash [\![BL_2]\!]_K \\
&\qquad \text{by def. of } [\![\ ]\!].
\end{aligned}$$

□

Proof *(Proposition 1).* Let K be a Kripke structure, φ be a state formula of CTRL, and δ be a propositional environment. We proceed by structural induction on φ.

- $\varphi ::= p$:

$$\begin{aligned}
[\![t(p)]\!]_K\delta &= [\![\langle X, \{X \overset{\mu}{=} p\}\rangle]\!]_K\delta && \text{by def. of } t \\
&= ([\![\{X \overset{\mu}{=} p\}]\!]_K\delta)(X) && \text{by def. of } [\![\ ]\!] \\
&= [\![p]\!]_K\delta && \text{by def. of } [\![\ ]\!].
\end{aligned}$$

- $\varphi ::= \varphi_1 \vee \varphi_2$ (similarly for $\varphi ::= \varphi_1 \wedge \varphi_2$):

$$
\begin{aligned}
[\![t(\varphi_1 \vee \varphi_2)]\!]_K\delta &= [\![\langle X, \{X \stackrel{\mu}{=} t_X(\varphi_1) \vee t_X(\varphi_2)\}.(t_{BL}(\varphi_1); t_{BL}(\varphi_2))\rangle]\!]_K\delta \\
&\qquad \text{by def. of } t \\
&= ([\![\{X \stackrel{\mu}{=} t_X(\varphi_1) \vee t_X(\varphi_2)\}.(t_{BL}(\varphi_1); t_{BL}(\varphi_2))]\!]_K\delta)(X) \\
&\qquad \text{by def. of } [\![\ ]\!] \\
&= ([\![\{X \stackrel{\mu}{=} t_X(\varphi_1) \vee t_X(\varphi_2)\}]\!]_K(\delta \oslash [\![t_{BL}(\varphi_1); t_{BL}(\varphi_2)]\!]_K\delta) \\
&\quad \oslash([\![t_{BL}(\varphi_1); t_{BL}(\varphi_2)]\!]_K\delta))(X) \\
&\qquad \text{by def. of } [\![\ ]\!] \\
&= ([\![\{X \stackrel{\mu}{=} t_X(\varphi_1) \vee t_X(\varphi_2)\}]\!]_K(\delta \oslash [\![t_{BL}(\varphi_1); t_{BL}(\varphi_2)]\!]_K\delta))(X) \\
&= [\![t_X(\varphi_1)]\!]_K(\delta \oslash [\![t_{BL}(\varphi_1)]\!]_K\delta \oslash [\![t_{BL}(\varphi_2)]\!]_K\delta) \cup \\
&\quad [\![t_X(\varphi_2)]\!]_K(\delta \oslash [\![t_{BL}(\varphi_1)]\!]_K\delta \oslash [\![t_{BL}(\varphi_2)]\!]_K\delta) \\
&= [\![t_X(\varphi_1)]\!]_K([\![t_{BL}(\varphi_1)]\!]_K\delta) \cup [\![t_X(\varphi_2)]\!]_K([\![t_{BL}(\varphi_2)]\!]_K\delta) \\
&\qquad \text{by Lemma 2} \\
&= [\![t(\varphi_1)]\!]_K\delta \cup [\![t(\varphi_2)]\!]_K\delta \\
&\qquad \text{by def. of } t \\
&= [\![\varphi_1]\!]_K\delta \cup [\![\varphi_2]\!]_K\delta \\
&\qquad \text{by ind. hyp.} \\
&= [\![\varphi_1 \vee \varphi_2]\!]_K\delta \\
&\qquad \text{by def. of } [\![\ ]\!].
\end{aligned}
$$

- $\varphi ::= \mathsf{EF}_\rho\varphi$ (similarly for $\varphi ::= \mathsf{AF}_\rho\varphi \mid \mathsf{EG}_\rho\varphi \mid \mathsf{AG}_\rho\varphi$):

$$
\begin{aligned}
[\![t(\mathsf{EF}_\rho\varphi)]\!]_K\delta &= [\![\langle X, \{X \stackrel{\mu}{=} \mathsf{EF}_\rho t_X(\varphi)\}.t_{BL}(\varphi)\rangle]\!]_K\delta \\
&\qquad \text{by def. of } t \\
&= ([\![t_{BL}(\varphi)]\!]_K\delta \oslash [\![\{X \stackrel{\mu}{=} \mathsf{EF}_\rho t_X(\varphi)\}]\!]_K([\![t_{BL}(\varphi)]\!]_K\delta))(X) \\
&\qquad \text{by def. of } [\![\ ]\!] \\
&= ([\![\{X \stackrel{\mu}{=} \mathsf{EF}_\rho t_X(\varphi)\}]\!]_K([\![t_{BL}(\varphi)]\!]_K\delta))(X) \\
&= [\![\{\mathsf{EF}_\rho t_X(\varphi)\}]\!]_K([\![t_{BL}(\varphi)]\!]_K\delta) \\
&\qquad \text{by def. of } [\![\ ]\!] \\
&= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\pi_{0,i} \models_K \rho \wedge \\
&\quad \pi_i \in [\![t_X(\varphi)]\!]_K([\![t_{BL}(\varphi)]\!]_K\delta)\} \\
&\qquad \text{by def. of } [\![\ ]\!] \\
&= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\pi_{0,i} \models_K \rho \wedge \\
&\quad \pi_i \in [\![\varphi]\!]_K\delta\} \\
&\qquad \text{by ind. hyp.} \\
&= [\![\mathsf{EF}_\rho\varphi]\!]_K\delta \\
&\qquad \text{by def. of } [\![\ ]\!].
\end{aligned}
$$

- $\varphi ::= \mathsf{EF}^\infty_\rho$ (similarly for $\varphi ::= \mathsf{AG}^\neg_\rho$):

$$
\begin{aligned}
[\![t(\mathsf{EF}^\infty_\rho)]\!]_K\delta &= [\![\langle X, \{X \stackrel{\nu}{=} \mathsf{EF}_\rho X\}\rangle]\!]_K \qquad \text{by def. of } t. \\
&= ([\![\{X \stackrel{\nu}{=} \mathsf{EF}_\rho X\}]\!]_K)(X) = \nu\Phi \quad \text{by def. of } [\![\ ]\!],
\end{aligned}
$$

where $\Phi : 2^S \to 2^S$, $\Phi(U) = [\![\mathsf{EF}_\rho X]\!]_K[U/X]$. Note that the δ environment is omitted in the definition of Φ because the equation block $\{X \stackrel{\nu}{=} \mathsf{EF}_\rho X\}$ is closed.

The lattice $\langle 2^S, \emptyset, S, \cap, \cup\rangle$ being finite, the maximal fixed point $\nu\Phi$ has also the following iterative characterization [80]:

$$\nu\Phi = \bigcap_{j\geq 0} \Phi^j(S), \quad \text{where} \quad \Phi^0(S) = S, \quad \Phi^j(S) = [\![\mathsf{EF}_\rho X]\!]_K[\Phi^{j-1}(S)/X].$$

Intuitively, the terms $\Phi^j(S)$ contain those states from which there is an outoing sequence having a prefix that matches ρ^j:

$$\Phi^j(S) = \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\pi_{0,i} \models_K \rho^j\}$$

This can be easily shown by induction on j. For $j = 0$, we take $i = 0$ (empty prefix). For the inductive step, we have:

$$\begin{aligned}
\Phi^{j+1}(S) &= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\pi_{0,i} \models_K \rho \wedge \pi_i \in \Phi^j(S)\} \\
&\qquad \text{by def. of } \Phi \\
&= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0. \\
&\qquad\qquad \pi_{0,i} \models_K \rho \wedge \exists \pi' \in Path_K(\pi_i).\exists l \geq 0.\pi'_{0,l} \models_K \rho^j\} \\
&\qquad \text{by ind. hyp.} \\
&= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\pi_{0,i} \models_K \rho^{j+1}\} \\
&\qquad \text{repl. } i \text{ by } i + l.
\end{aligned}$$

To show that $[\![\mathsf{EF}^\infty_\rho]\!]_K \subseteq \nu\Phi$, let $s \in [\![\mathsf{EF}^\infty_\rho]\!]_K$ and $j \geq 0$. From the definition of EF^∞_ρ, there exists $\pi \in Path_K(s)$ and $i \geq 0$ such that $\pi_{0,i} \models_K \rho^j$, which implies $s \in \Phi^j(S)$. Since this holds for every $j \geq 0$, it means that $s \in \bigcap_{j\geq 0} \Phi^j(S)$, i.e., $s \in \nu\Phi$.

To show that $\nu\Phi \subseteq [\![\mathsf{EF}^\infty_\rho]\!]_K$, let $s \in \nu\Phi$. Since $\nu\Phi$ is a fixed point of Φ, we have:

$$\begin{aligned}
\nu\Phi &= \Phi(\nu\Phi) = [\![\mathsf{EF}_\rho X]\!]_K[\nu\Phi/X] \\
&= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\pi_{0,i} \models_K \rho \wedge \pi_i \in \nu\Phi\}.
\end{aligned}$$

Based on this, we construct the following path:

$$\overline{\pi} = \overline{\pi}_{i_0} \to \cdots \to \overline{\pi}_{i_1} \to \cdots \to \overline{\pi}_{i_2} \to \cdots \to \overline{\pi}_{i_j} \cdots$$

where $\overline{\pi}_{i_j} \in \nu\Phi$ for every $j \geq 0$, $i_0 = 0$, $\overline{\pi}_{i_0} = s$, and the intervals $\overline{\pi}_{i_j} \to \cdots \to \overline{\pi}_m \to \cdots \to \overline{\pi}_{i_{j+1}}$ are defined as follows. Since $\overline{\pi}_{i_j} \in \nu\Phi$, according to the equation above, there exists $\pi \in Path_K(\overline{\pi}_{i_j})$ and $l \geq 0$ such that $\pi_{0,l} \models_K \rho$ and $\pi_l \in \nu\Phi$. We take $i_{j+1} = i_j + l$ and for each $m \in [i_j, i_{j+1}]$, $\overline{\pi}_m = \pi_{m-i_j}$. The infinite path $\overline{\pi}$ is such that for every $j \geq 0$, there exists $i' = i_j$ such that $\overline{\pi}_{0,i'} \models_K \rho^j$, and therefore $s \in [\![\mathsf{EF}^\infty_\rho]\!]_K$.

□

A.2 Translation from RESs to MESs

Some additional definitions and lemmas are needed in order to prove the translation. Let $B = \{X_i \stackrel{\sigma}{=} \varphi_i\}_{1\leq i\leq n}$ be an equation block and $\Phi_\delta : (2^S)^n \to (2^S)^n$, $\Phi_\delta(U_1, ..., U_n) = \langle [\![\varphi_i]\!]_K(\delta \oslash [U_1/X_1, ..., U_n/X_n])\rangle_{1\leq i\leq n}$ be its associated functional in the context of a Kripke structure K and an environment δ. For a given $l \in [1, n]$, the *projection* of Φ_δ on the equations $[l, n]$, noted $\Phi^{l,n} : (2^S)^{n-l+1} \to (2^S)^{n-l+1}$, is defined as follows: $\Phi^{l,n}_\delta(U_l, ..., U_n) = \langle [\![\varphi_j]\!]_K(\delta \oslash [U_l/X_l, ..., U_n/X_n])\rangle_{l\leq j\leq n}$. Similarly, the projection of a value $\langle U_1, ..., U_n\rangle \in (2^S)^n$ on the fields $[l, n]$ is defined as $\langle U_1, ..., U_n\rangle_{[l,n]} = \langle U_l, ..., U_n\rangle$.

A.2.1 Operators EF_ρ and AG_ρ

Lemma 4 *Let $B = \{X_i \stackrel{\sigma}{=} \varphi_i\}_{1\leq i\leq n}$ be an equation block, K be a Kripke structure, δ be an environment, and $\Phi_\delta : (2^S)^n \to (2^S)^n$ be the functional associated to B, K, and δ. Then, for all $l \in [1, n]$:*

$$\sigma\Phi^{l,n}_{\delta\oslash[(\sigma\Phi_\delta)_1/X_1,...,(\sigma\Phi_\delta)_{l-1}/X_{l-1}]} = \langle(\sigma\Phi_\delta)_l, ..., (\sigma\Phi_\delta)_n\rangle$$

where $\Phi^{l,n}_\delta : (2^S)^{n-l+1} \to (2^S)^{n-l+1}$ is the projection of Φ_δ on the equations $[l, n]$.

Proof Let B, K, δ, and l as stated in the hypothesis. We show the equality by double inclusion, only for $\sigma = \mu$, the proof for the case $\sigma = \nu$ being symmetric.
Inclusion "$\sqsupseteq$": By definition of fixed points we have $\mu\Phi_\delta = \Phi_\delta(\mu\Phi_\delta)$, meaning that for all $l \leq j \leq n$:

$$\begin{array}{l}(\mu\Phi_\delta)_j = [\![\varphi_j]\!]_K(\delta \oslash [(\mu\Phi_\delta)_1/X_1, ..., (\mu\Phi_\delta)_n/X_n]) = \\ [\![\varphi_j]\!]_K((\delta \oslash [(\mu\Phi_\delta)_1/X_1, ..., (\mu\Phi_\delta)_{l-1}/X_{l-1}]) \oslash [(\mu\Phi_\delta)_l/X_l, ..., (\mu\Phi_\delta)_n/X_n])\end{array}$$

This in turn means that:

$$\Phi^{l,n}_{\delta\oslash[(\sigma\Phi_\delta)_1/X_1,...,(\sigma\Phi_\delta)_{l-1}/X_{l-1}]}((\mu\Phi_\delta)_l, ..., (\mu\Phi_\delta)_n) = \langle(\mu\Phi_\delta)_l, ..., (\mu\Phi_\delta)_n\rangle$$

i.e., $\langle(\mu\Phi_\delta)_l, ..., (\mu\Phi_\delta)_n\rangle$ is a fixed point of $\Phi^{l,n}_{\delta\oslash[(\sigma\Phi_\delta)_1/X_1,...,(\sigma\Phi_\delta)_{l-1}/X_{l-1}]}$, and therefore it is greater than the least fixed point of this functional:

$$\mu\Phi^{l,n}_{\delta\oslash[(\sigma\Phi_\delta)_1/X_1,...,(\sigma\Phi_\delta)_{l-1}/X_{l-1}]} \sqsubseteq \langle(\mu\Phi_\delta)_l, ..., (\mu\Phi_\delta)_n\rangle.$$

Inclusion "$\sqsupseteq$": We use the iterative characterization [80] of $\mu\Phi_\delta$ on the finite lattice $\langle 2^{S^n}, \emptyset, S^n, \sqcap, \sqcup\rangle$ (the operations $\sqcap$ and $\sqcup$ are the pairwise extensions of $\cap$ and $\cup$):

$$\mu\Phi_\delta = \bigcup_{k\geq 0} \Phi^k_\delta(\emptyset^n), \quad \text{where} \quad \Phi^0_\delta(\emptyset^n) = \emptyset^n, \quad \Phi^{k+1}_\delta(\emptyset^n) = \Phi_\delta(\Phi^k_\delta(\emptyset^n)).$$

We show, by induction on k, that $(\Phi_\delta^k(\emptyset^n))_{[l,n]} \sqsubseteq \mu\Phi^{l,n}_{\delta\oslash[(\sigma\Phi_\delta)_1/X_1,\ldots,(\sigma\Phi_\delta)_{l-1}/X_{l-1}]}$.
Base step. $(\Phi_\delta^0(\emptyset^n))_{[l,n]} = (\emptyset^n)_{[l,n]} = \emptyset^{n-l+1} \sqsubseteq \mu\Phi^{l,n}_{\delta\oslash[(\sigma\Phi_\delta)_1/X_1,\ldots,(\sigma\Phi_\delta)_{l-1}/X_{l-1}]}$.
Inductive step. We have:

$$
\begin{aligned}
(\Phi_\delta^{k+1}(\emptyset^n))_{[l,n]} &= (\Phi_\delta(\Phi_\delta^k(\emptyset^n)))_{[l,n]} \\
&= \langle [\![\varphi_j]\!]_K(\delta \oslash [(\Phi_\delta^k(\emptyset^n))_1/X_1, ..., (\Phi_\delta^k(\emptyset^n))_n/X_n])\rangle_{l\le j\le n} \\
&\qquad \text{by def. of } \Phi \\
&= \langle [\![\varphi_j]\!]_K((\delta \oslash [(\Phi_\delta^k(\emptyset^n))_1/X_1, ..., (\Phi_\delta^k(\emptyset^n))_{l-1}/X_{l-1}]) \oslash \\
&\qquad [(\Phi_\delta^k(\emptyset^n))_l/X_l, ..., (\Phi_\delta^k(\emptyset^n))_n/X_n])\rangle_{l\le j\le n} \\
&\sqsubseteq \langle [\![\varphi_j]\!]_K((\delta \oslash [(\mu\Phi_\delta)_1/X_1, ..., (\mu\Phi_\delta)_{l-1}/X_{l-1}]) \oslash \\
&\qquad [(\Phi_\delta^k(\emptyset^n))_l/X_l, ..., (\Phi_\delta^k(\emptyset^n))_n/X_n])\rangle_{l\le j\le n} \\
&\qquad \text{by monotonicity} \\
&= \Phi^{l,n}_{\delta\oslash[(\mu\Phi_\delta)_1/X_1,\ldots,(\mu\Phi_\delta)_{l-1}/X_{l-1}]}((\Phi_\delta^k(\emptyset^n))_l, ..., (\Phi_\delta^k(\emptyset^n))_n) \\
&\qquad \text{by def. of } \Phi^{l,n} \\
&= \Phi^{l,n}_{\delta\oslash[(\mu\Phi_\delta)_1/X_1,\ldots,(\mu\Phi_\delta)_{l-1}/X_{l-1}]}((\Phi_\delta^k(\emptyset^n))_{[l,n]}) \\
&\sqsubseteq \Phi^{l,n}_{\delta\oslash[(\mu\Phi_\delta)_1/X_1,\ldots,(\mu\Phi_\delta)_{l-1}/X_{l-1}]}(\mu\Phi^{l,n}_{\delta\oslash[(\sigma\Phi_\delta)_1/X_1,\ldots,(\sigma\Phi_\delta)_{l-1}/X_{l-1}]}) \\
&\qquad \text{by ind. hyp.} \\
&= \mu\Phi^{l,n}_{\delta\oslash[(\sigma\Phi_\delta)_1/X_1,\ldots,(\sigma\Phi_\delta)_{l-1}/X_{l-1}]} \\
&\qquad \text{by def. of } \mu.
\end{aligned}
$$

Thus, $(\mu\Phi_\delta)_{[l,n]} = (\bigcup_{k\ge 0}\Phi_\delta^k(\emptyset^n))_{[l,n]} = \bigcup_{k\ge 0}(\Phi_\delta^k(\emptyset^n))_{[l,n]} \sqsubseteq \mu\Phi^{l,n}_{\delta\oslash[(\sigma\Phi_\delta)_1/X_1,\ldots,(\sigma\Phi_\delta)_{l-1}/X_{l-1}]}$, which concludes the proof. □

The following lemma allows to replace an equation of a block by a set of equations, provided that the interpretation of the variable in the left-hand side of the equation remains unchanged in the original and the substituting block w.r.t. all environments.

Lemma 5 (Substitution) *Let $B = \{X_i \overset{\sigma}{=} \varphi_i\}_{1\le i\le n}$ be an equation block, and let $\{X_n \overset{\sigma}{=} \psi_n, Y_j \overset{\sigma}{=} \psi_j\}_{n<j\le m}$ be another block suitable for the substitution of the equation $X_n \overset{\sigma}{=} \varphi_n$ such that $([\![\{X_n \overset{\sigma}{=} \varphi_n\}]\!]_K\delta)(X_n) = ([\![\{X_n \overset{\sigma}{=} \psi_n, Y_j \overset{\sigma}{=} \psi_j\}_{n<j\le m}]\!]_K\delta)(X_n)$ for any Kripke structure K and environment δ. Then:*

$$
\begin{aligned}
&([\![\{X_i \overset{\sigma}{=} \varphi_i\}_{1\le i\le n}[X_n \overset{\sigma}{=} \varphi_n := X_n \overset{\sigma}{=} \psi_n, Y_j \overset{\sigma}{=} \psi_j]_{n<j\le m}]\!]_K\delta)(X_i) = \\
&([\![\{X_i \overset{\sigma}{=} \varphi_i\}_{1\le i\le n}]\!]_K\delta)(X_i)
\end{aligned}
$$

for all $i \in [1, n]$ and for any K, δ.

Proof We show the lemma for $\sigma = \mu$, the proof for the case $\sigma = \nu$ being symmetric. Let $\Phi_\delta^{1,m} : (2^S)^m \to (2^S)^m$ be the functional associated to the substituted equation block, defined as follows:

$$
\begin{aligned}
&\Phi_\delta^{1,m}(U_1, ..., U_n, W_{n+1}, ..., W_n) = \\
&\quad \langle [\![\varphi_i]\!]_K(\delta \oslash [U_1/X_1, ..., U_n/X_n, W_{n+1}/Y_{n+1}, ..., W_m/Y_m]), \\
&\quad\ [\![\psi_j]\!]_K(\delta \oslash [U_1/X_1, ..., U_n/X_n, W_{n+1}/Y_{n+1}, ..., W_m/Y_m])\rangle_{1\le i<n, n\le j\le m}
\end{aligned}
$$

We first show that $\langle(\mu\Phi_\delta^{1,m})_1, ..., (\mu\Phi_\delta^{1,m})_n\rangle$ is a fixed point of the functional Φ_δ associated to B and δ. From the definition of $\mu\Phi_\delta^{1,m}$, it follows that $[\![\varphi_i]\!]_K(\delta \oslash [(\mu\Phi_\delta^{1,m})_1/X_1, ..., (\mu\Phi_\delta^{1,m})_n/X_n, (\mu\Phi_\delta^{1,m})_{n+1}/Y_{n+1}, ...$ $(\mu\Phi_\delta^{1,m})_i$ for all $i \in [1, n-1]$. The suitability condition $\bigcup_{i=1}^n fv(\varphi_i) \cap \{Y_{n+1}, ..., Y_m\} = \emptyset$ implies that all formulas φ_i for $i \in [1, n-1]$ depend only upon $X_1, ..., X_n$ and therefore $[\![\varphi_i]\!]_K(\delta \oslash [(\mu\Phi_\delta^{1,m})_1/X_1, ..., (\mu\Phi_\delta^{1,m})_n/X_n]) = (\mu\Phi_\delta^{1,m})_i$. To show that this equality also holds for $i = n$, we apply Lemma 4 for $l = n$ on the substituted block and we obtain:

$$\mu\Phi^{n,m}_{\delta \oslash [(\mu\Phi_\delta^{1,m})_1/X_1, ..., (\mu\Phi_\delta^{1,m})_{n-1}/X_{n-1}]} = \langle(\mu\Phi_\delta^{1,m})_n, ..., (\mu\Phi_\delta^{1,m})_m\rangle$$

where $\Phi_\delta^{n,m} : (2^S)^{m-n+1} \to (2^S)^{m-n+1}$ is the projection of $\Phi_\delta^{1,m}$ on the equations $[n, m]$. From the hypothesis of the lemma and the definition of the interpretation $[\![\{X_n \stackrel{\sigma}{=} \psi_n, Y_j \stackrel{\sigma}{=} \psi_j\}_{n<j\leq m}]\!]_K\delta$, this implies:

$$([\![\{X_n \stackrel{\sigma}{=} \varphi_n\}]\!]_K(\delta \oslash [(\mu\Phi_\delta^{1,m})_1/X_1, ..., (\mu\Phi_\delta^{1,m})_{n-1}/X_{n-1}]))(X_n) = (\mu\Phi_\delta^{1,m})_n$$

or, according to the definition of $[\![\{X_n \stackrel{\sigma}{=} \varphi_n\}]\!]_K\delta$:

$$\mu\Phi^{n}_{\delta \oslash [(\mu\Phi_\delta^{1,m})_1/X_1, ..., (\mu\Phi_\delta^{1,m})_{n-1}/X_{n-1}]} = (\mu\Phi_\delta^{1,m})_n$$

where $\Phi_\delta^n : 2^S \to 2^S$, $\Phi_\delta^n(U) = [\![\varphi_n]\!]_K(\delta \oslash [U/X_n])$. Since $(\mu\Phi_\delta^{1,m})_n$ is by definition a fixed point of $\Phi^{n}_{\delta \oslash [(\mu\Phi_\delta^{1,m})_1/X_1, ..., (\mu\Phi_\delta^{1,m})_{n-1}/X_{n-1}]}$, this means:

$$\begin{array}{c}([\![\varphi_n]\!]_K((\delta \oslash [(\mu\Phi_\delta^{1,m})_1/X_1, ..., (\mu\Phi_\delta^{1,m})_{n-1}/X_{n-1}]) \oslash \\ [(\mu\Phi_\delta^{1,m})_n/X_n]))((\mu\Phi_\delta^{1,m})_n) = (\mu\Phi_\delta^{1,m})_n\end{array}$$

i.e.,

$$([\![\varphi_n]\!]_K(\delta \oslash [(\mu\Phi_\delta^{1,m})_1/X_1, ..., (\mu\Phi_\delta^{1,m})_n/X_n]))((\mu\Phi_\delta^{1,m})_n) = (\mu\Phi_\delta^{1,m})_n.$$

Therefore, $\langle(\mu\Phi_\delta^{1,m})_i\rangle_{1\leq i\leq n}$ is a fixed point of Φ_δ.

It remains to show that this is indeed the minimal fixed point of Φ_δ. Since the lattice $\langle 2^{S^m}, \emptyset, S^m, \sqcap, \sqcup\rangle$ is finite (the operations $\sqcap$ and $\sqcup$ being the pairwise extensions of $\cap$ and $\cup$), the minimal fixed point $\mu\Phi_\delta^{1,m}$ also has an iterative characterization [80]:

$$\mu\Phi_\delta^{1,m} = \bigcup_{k\geq 0} (\Phi_\delta^{1,m})^k(\emptyset^m)$$

where $(\Phi_\delta^{1,m})^0(\emptyset^m) = \emptyset^m$, $(\Phi_\delta^{1,m})^{k+1}(\emptyset^m) = \Phi_\delta^{1,m}((\Phi_\delta^{1,m})^k(\emptyset^m))$.

We show, by induction on k, that $((\Phi_\delta^{1,m})^k(\emptyset^m))_i \subseteq (\mu\Phi_\delta)_i$ for all $i \in [1, n]$ and $k \geq 0$. Let $i \in [1, n]$.

Base step. $((\Phi_\delta^{1,m})^0(\emptyset^m))_i = \emptyset \subseteq (\mu\Phi_\delta)_i$.

Inductive step. For $i \in [1, n-1]$, we have:

$$
\begin{aligned}
((\Phi_\delta^{1,m})^{k+1}(\emptyset^m))_i &= (\Phi_\delta^{1,m}((\Phi_\delta^{1,m})^k(\emptyset^m)))_i \\
&= [\![\varphi_i]\!]_K(\delta \oslash [((\Phi_\delta^{1,m})^k(\emptyset^m))_1/X_1, ..., ((\Phi_\delta^{1,m})^k(\emptyset^m))_m/Y_m]) \\
&\qquad \text{by def. of } [\![\ ]\!] \\
&= [\![\varphi_i]\!]_K(\delta \oslash [((\Phi_\delta^{1,m})^k(\emptyset^m))_1/X_1, ..., ((\Phi_\delta^{1,m})^k(\emptyset^m))_n/X_n]) \\
&\qquad \text{by suitability} \\
&\subseteq [\![\varphi_i]\!]_K(\delta \oslash [(\mu\Phi_\delta)_1/X_1, ..., (\mu\Phi_\delta)_n/X_n]) \\
&\qquad \text{by ind. hyp.} \\
&= (\mu\Phi_\delta)_i \\
&\qquad \text{by def. of } \mu\Phi_\delta.
\end{aligned}
$$

For $i = n$, we have:

$$
\begin{aligned}
((\Phi_\delta^{1,m})^{k+1}(\emptyset^m))_n &= [\![\psi_n]\!]_K(\delta \oslash [((\Phi_\delta^{1,m})^k(\emptyset^m))_1/X_1, ..., ((\Phi_\delta^{1,m})^k(\emptyset^m))_m/Y_m]) \\
&\subseteq [\![\psi_n]\!]_K(\delta \oslash [((\Phi_\delta^{1,m})^k(\emptyset^m))_1/X_1, ..., ((\Phi_\delta^{1,m})^k(\emptyset^m))_{n-1}/X_{n-1}, \\
&\qquad\qquad (\mu\Phi_\delta^{1,m})_n/X_n, ..., (\mu\Phi_\delta^{1,m})_m/Y_m]) \\
&\qquad \text{by def. } \mu\Phi_\delta^{1,m} \\
&= ([\![\{X_n \overset{\mu}{=} \psi_n, Y_j \overset{\mu}{=} \psi_j\}_{n<j\leq m}]\!]_K \\
&\quad (\delta \oslash [((\Phi_\delta^{1,m})^k(\emptyset^m))_1/X_1, ..., ((\Phi_\delta^{1,m})^k(\emptyset^m))_{n-1}/X_{n-1}]))(X_n) \\
&\qquad \text{by def. of } [\![\ ]\!] \\
&= ([\![X_n \overset{\mu}{=} \varphi_n]\!]_K \\
&\quad (\delta \oslash [((\Phi_\delta^{1,m})^k(\emptyset^m))_1/X_1, ..., ((\Phi_\delta^{1,m})^k(\emptyset^m))_{n-1}/X_{n-1}]))(X_n) \\
&\qquad \text{by hyp.} \\
&= \mu\Phi^n_{\delta\oslash[((\Phi_\delta^{1,m})^k(\emptyset^m))_1/X_1,...,((\Phi_\delta^{1,m})^k(\emptyset^m))_{n-1}/X_{n-1}]} \\
&\qquad \text{by def. of } [\![\ ]\!] \\
&\subseteq \mu\Phi^n_{\delta\oslash[(\mu\Phi_\delta)_1/X_1,...,(\mu\Phi_\delta)_{n-1}/X_{n-1}]} \\
&\qquad \text{by ind. hyp.} \\
&= (\mu\Phi_\delta)_n \\
&\qquad \text{by Lemma 4.}
\end{aligned}
$$

The last application of Lemma 4 above considers the block $B = \{X_i \overset{\sigma}{=} \varphi_i\}_{1\leq i\leq n}$ and takes $l = n$. This concludes the proof that $\langle(\mu\Phi_\delta^{1,m})_i\rangle_{1\leq i\leq n}$ is the least fixed point of Φ_δ. □

Lemma 5 allows to prove the correctness of a substitution by focusing only on the equations involved in the substitution, as illustrated in the proof below.

Proof *(Proposition 2).* Let K be a Kripke structure, $B_1 = \{X_i \overset{\mu}{=} \varphi_i\}_{1\leq i\leq n}$ and $B_2 = \{X_i \overset{\nu}{=} \varphi_i\}_{1\leq i\leq n}$ two equation blocks, and δ a propositional environment as stated in the hypothesis. We show the proposition only for blocks of type B_1 and the substitutions in the upper part of Figure 3.5, the other cases being dual.

- Substitution $X \overset{\mu}{=} \mathsf{EF}_{\rho_1.\rho_2}Y := X \overset{\mu}{=} \mathsf{EF}_{\rho_1}Z, Z \overset{\mu}{=} \mathsf{EF}_{\rho_2}Y$. It is sufficient to show that this

substitution satisfies the condition in the hypothesis of Lemma 5:

$$([\![\{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1.\rho_2} Y\}]\!]_K \delta)(X) = ([\![\{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} Z, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho_2} Y\}]\!]_K \delta)(X).$$

By applying the definition of $[\![\ ]\!]$ and simple properties about substitution of variables in a RES, we obtain:

$$\begin{aligned}
&([\![\{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1.\rho_2} Y\}]\!]_K \delta)(X) \\
&= [\![\mathsf{EF}_{\rho_1.\rho_2} Y]\!]_K \delta \\
&= \{s \in S \mid \exists \pi \in Path_K(s). \exists i \geq 0. \pi_{0,i} \models_K \rho_1.\rho_2 \wedge \pi_i \in [\![Y]\!]_K\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s). \exists i \geq 0. \exists k \in [0,i]. \\
&\qquad \pi_{0,k} \models_K \rho_1 \wedge \pi_{k,i} \models_K \rho_2 \wedge \pi_i \in [\![Y]\!]_K\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s). \exists k \geq 0. \\
&\qquad \pi_{0,k} \models_K \rho_1 \wedge \exists i \geq k. \pi_{k,i} \models_K \rho_2 \wedge \pi_i \in [\![Y]\!]_K\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s). \exists k \geq 0. \pi_{0,k} \models_K \rho_1 \wedge \exists \pi' \in Path_K(\pi_k). \exists i \geq k. \\
&\qquad \pi'_{k,i} \models_K \rho_2 \wedge \pi'_i \in [\![Y]\!]_K\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s). \exists k \geq 0. \pi_{0,k} \models_K \rho \wedge \pi_k \in [\![\mathsf{EF}_{\rho_2} Y]\!]_K\} \\
&= [\![\mathsf{EF}_{\rho_1} \mathsf{EF}_{\rho_2} Y]\!]_K \delta \\
&= ([\![\{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} \mathsf{EF}_{\rho_2} Y\}]\!]_K \delta)(X) \\
&= ([\![\{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} Z, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho_2} Y\}]\!]_K \delta)(X).
\end{aligned}$$

- Substitution $X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1|\rho_2} Y := X \stackrel{\mu}{=} Z \vee U, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} Y, U \stackrel{\mu}{=} \mathsf{EF}_{\rho_2} Y$. As above, it is sufficient to show that:

$$\begin{aligned}
&([\![\{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1|\rho_2} Y\}]\!]_K \delta)(X) = \\
&([\![\{X \stackrel{\mu}{=} Z \vee U, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} Y, U \stackrel{\mu}{=} \mathsf{EF}_{\rho_2} Y\}]\!]_K \delta)(X).
\end{aligned}$$

By applying the definition of $[\![\ ]\!]$ and simple properties about substitution of variables in a RES, we obtain:

$$\begin{aligned}
&([\![\{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1|\rho_2} Y\}]\!]_K \delta)(X) \\
&= [\![\mathsf{EF}_{\rho_1|\rho_2} Y]\!]_K \delta \\
&= \{s \in S \mid \exists \pi \in Path_K(s). \exists i \geq 0. \pi_{0,i} \models_K \rho_1|\rho_2 \wedge \pi_i \in [\![Y]\!]_K \delta\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s). \exists i \geq 0. (\pi_{0,i} \models_K \rho_1 \vee \pi_{0,i} \models_K \rho_2) \wedge \\
&\qquad \pi_i \in [\![Y]\!]_K \delta\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s). \exists i \geq 0. ((\pi_{0,i} \models_K \rho_1 \wedge \pi_i \in [\![Y]\!]_K \delta) \vee \\
&\qquad (\pi_{0,i} \models_K \rho_2 \wedge \pi_i \in [\![Y]\!]_K \delta))\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s). \exists i \geq 0. \pi_{0,i} \models_K \rho_1 \wedge \pi_i \in [\![Y]\!]_K \delta \vee \\
&\qquad \exists \pi \in Path_K(s). \exists i \geq 0. \pi_{0,i} \models_K \rho_2 \wedge \pi_i \in [\![Y]\!]_K \delta\} \\
&= [\![\mathsf{EF}_{\rho_1} Y \vee \mathsf{EF}_{\rho_2} Y]\!]_K \delta \\
&= ([\![\{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} Y \vee \mathsf{EF}_{\rho_2} Y\}]\!]_K \delta)(X) \\
&= ([\![\{X \stackrel{\mu}{=} Z \vee U, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} Y, U \stackrel{\mu}{=} \mathsf{EF}_{\rho_2} Y\}]\!]_K \delta)(X).
\end{aligned}$$

- Substitution $X \stackrel{\mu}{=} \mathsf{EF}_{\rho^*} Y := X \stackrel{\mu}{=} Y \vee Z, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho} X$. As above, it is sufficient to show that:

$$(\llbracket \{X \stackrel{\mu}{=} \mathsf{EF}_{\rho^*} Y\} \rrbracket_K \delta)(X) = (\llbracket \{X \stackrel{\mu}{=} Y \vee Z, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho} X\} \rrbracket_K \delta)(X).$$

Let $A = (\llbracket \{X \stackrel{\mu}{=} \mathsf{EF}_{\rho^*} Y\} \rrbracket_K \delta)(X)$. We have:

$(\llbracket \{X \stackrel{\mu}{=} \mathsf{EF}_{\rho^*} Y\} \rrbracket_K \delta)(X) =$ by def. of $\llbracket\ \rrbracket$
$\llbracket \mathsf{EF}_{\rho^*} Y \rrbracket_K \delta =$
$\{s \in S \mid \exists \pi \in Path_K(s). \exists i \geq 0. \pi_{0,i} \models_K \rho^* \wedge \pi_i \in \llbracket Y \rrbracket_K \delta\} =$
$\{s \in S \mid \exists \pi \in Path_K(s). \exists i \geq 0. \exists k \geq 0. \pi_{0,i} \models_K \rho^k \wedge \pi_i \in \delta(Y)\}.$

Let $B = (\llbracket \{X \stackrel{\mu}{=} Y \vee Z, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho} X\} \rrbracket_K \delta)(X)$. We have:

$(\llbracket \{X \stackrel{\mu}{=} Y \vee Z, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho} X\} \rrbracket_K \delta)(X) =$ by subst. on Y
$(\llbracket \{X \stackrel{\mu}{=} Y \vee \mathsf{EF}_{\rho} X\} \rrbracket_K \delta)(X) = \mu \Phi_\delta$

where the functional $\Phi_\delta : 2^S \to 2^S$ is defined as follows:

$$\begin{aligned}\Phi_\delta(U) &= \llbracket Y \vee \mathsf{EF}_{\rho} X \rrbracket_K (\delta \oslash [U/X]) \\ &= \llbracket Y \rrbracket_K (\delta \oslash [U/X]) \cup \llbracket \mathsf{EF}_{\rho} X \rrbracket_K (\delta \oslash [U/X]) \\ &= \delta(Y) \cup \llbracket \mathsf{EF}_{\rho} X \rrbracket_K [U/X].\end{aligned}$$

The lattice $\langle 2^S, \emptyset, S, \cap, \cup \rangle$ being finite, the minimal fixed point $\mu \Phi_\delta$ has also the following iterative characterization [80]:

$$\mu \Phi_\delta = \bigcup_{k \geq 0} \Phi^k(\emptyset),$$

where $\Phi^0(\emptyset) = \emptyset$, $\Phi^{k+1}(\emptyset) = \delta(Y) \cup \llbracket \mathsf{EF}_{\rho} X \rrbracket_K [\Phi^k(\emptyset)/X]$.

Intuitively, $\Phi^{k+1}(\emptyset)$ contains those states having an outgoing sequence that matches ρ^j for some $j \in [0, k]$ and leads to a state in $\delta(Y)$:

$$\begin{aligned}\Phi^{k+1}(\emptyset) = \{s \in S \mid \exists \pi \in Path_K(s). \exists i \geq 0. \exists j \in [0, k]. \\ \pi_{0,i} \models_K \rho^j \wedge \pi_i \in \delta(Y)\}.\end{aligned}$$

This statement can be easily shown by induction on k.
Base step.

$$\begin{aligned}\Phi^1(\emptyset) &= \delta(Y) \cup \llbracket \mathsf{EF}_{\rho} X \rrbracket_K [\Phi^0(\emptyset)/X] \\ &= \delta(Y) \cup \llbracket \mathsf{EF}_{\rho} X \rrbracket_K [\emptyset/X] \\ &= \delta(Y) \\ &= \{s \in S \mid \exists \pi \in Path_K(s). \pi_{0,0} \models_K \rho^0 \wedge \pi_0 \in \delta(Y)\} \\ &\qquad \text{by choosing } i, j = 0.\end{aligned}$$

Inductive step.

$$
\begin{aligned}
\Phi^{k+2}(\emptyset) &= \Phi(\Phi^{k+1}(\emptyset)) \qquad \text{by def. of } \Phi \\
&= \delta(Y) \cup \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0. \\
&\qquad\qquad \pi_{0,i} \models_K \rho \wedge \pi_i \in \Phi^{k+1}(\emptyset)\} \qquad \text{by ind. hyp.} \\
&= \delta(Y) \cup \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\pi_{0,i} \models_K \rho \wedge \\
&\qquad\qquad \exists \pi' \in Path_K(\pi_i).\exists i' \geq 0.\exists j \in [0,k]. \\
&\qquad\qquad \pi'_{0,i'} \models_K \rho^j \wedge \pi'_{i'} \in \delta(Y)\} \\
&= \delta(Y) \cup \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\exists j \in [0,k]. \\
&\qquad\qquad \pi_{0,i} \models_K \rho^{j+1} \wedge \pi_i \in \delta(Y)\} \quad \text{repl. } i \text{ by } i+i' \\
&= \delta(Y) \cup \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\exists j \in [1,k+1]. \\
&\qquad\qquad \pi_{0,i} \models_K \rho^j \wedge \pi_i \in \delta(Y)\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\exists j \in [0,k+1]. \\
&\qquad\qquad \pi_{0,i} \models_K \rho^j \wedge \pi_i \in \delta(Y)\}.
\end{aligned}
$$

From the above statement, we obtain:

$$
\begin{aligned}
B &= \textstyle\bigcup_{k\geq 0} \Phi^k(\emptyset) = \Phi^0(\emptyset) \cup \bigcup_{k\geq 0} \Phi^{k+1}(\emptyset) = \bigcup_{k\geq 0} \Phi^{k+1}(\emptyset) \\
&= \textstyle\bigcup_{k\geq 0}\{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\exists j \in [0,k]. \\
&\qquad \pi_{0,i} \models_K \rho^j \wedge \pi_i \in \delta(Y)\} \\
&= \{s \in S \mid \exists k \geq 0.\exists \pi \in Path_K(s).\exists i \geq 0.\exists j \in [0,k]. \\
&\qquad \pi_{0,i} \models_K \rho^j \wedge \pi_i \in \delta(Y)\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\exists k \geq 0.\exists j \in [0,k]. \\
&\qquad \pi_{0,i} \models_K \rho^j \wedge \pi_i \in \delta(Y)\} \\
&= \{s \in S \mid \exists \pi \in Path_K(s).\exists i \geq 0.\exists k \geq 0.\pi_{0,i} \models_K \rho^k \wedge \pi_i \in \delta(Y)\} \\
&\qquad \text{choose } j = k \\
&= A.
\end{aligned}
$$

□

A.2.2 Operators AF_ρ and EG_ρ

Translation to guarded form

Proof *(Lemma 1).* Let K be a Kripke structure, $B = \{X_i \stackrel{\mu}{=} \varphi_i\}_{1\leq i\leq n}$ be an equation block and δ a propositional environment as stated in the hypothesis. It is sufficient to show that the absorption substitution satisfies the condition in the hypothesis of Lemma 5:

$$(\llbracket\{X \stackrel{\mu}{=} X \vee \varphi\}\rrbracket_K \delta)(X) = (\llbracket\{X \stackrel{\mu}{=} \varphi\}\rrbracket_K \delta)(X)$$

which amounts to show, applying the definition of $\llbracket\ \rrbracket$, that:

$$\mu\Phi_\delta = \llbracket\varphi\rrbracket_K \delta$$

where the functional $\Phi_\delta : 2^S \to 2^S$ is defined as $\Phi_\delta(U) = [\![X \vee \varphi]\!]_K(\delta \oslash [U/X]) = U \cup [\![\varphi]\!]_K\delta$. The lattice $\langle 2^S, \emptyset, S, \cap, \cup\rangle$ being finite, the minimal fixed point $\mu\Phi_\delta$ has also the following iterative characterization [80]:

$$\mu\Phi_\delta = \bigcup_{k\geq 0} \Phi^k(\emptyset), \quad \text{where} \quad \Phi^0_\delta(\emptyset) = \emptyset, \quad \Phi^{k+1}_\delta(\emptyset) = \Phi^k_\delta(\emptyset) \cup [\![\varphi]\!]_K\delta.$$

To obtain the desired equality, it is therefore sufficient to show that $\Phi^{k+1}_\delta(\emptyset) = [\![\varphi]\!]_K\delta$ for every $k \geq 0$. We proceed by induction on k.

Base step: $\Phi^1_\delta(\emptyset) = \Phi^0_\delta(\emptyset) \cup [\![\varphi]\!]_K\delta = [\![\varphi]\!]_K\delta$.

Inductive step:

$$\begin{aligned}\Phi^{k+1}_\delta(\emptyset) &= \Phi^k_\delta(\emptyset) \cup [\![\varphi]\!]_K\delta \quad \text{by def.}\\ &= [\![\varphi]\!]_K\delta \cup [\![\varphi]\!]_K\delta \quad \text{by ind. hyp.}\\ &= [\![\varphi]\!]_K\delta.\end{aligned}$$

□

Determinization

Several definitions and lemmas are needed in order to prove Proposition 3. Consider a Kripke structure K and the following potentiality RES:

$$\left\{ X_i \stackrel{\mu}{=} \bigvee_{j=1}^{n} (h_{ij} \wedge \mathsf{EF}_{\rho_{ij}} X_j) \vee (h_i \wedge Y) \right\}_{1\leq i\leq n} \qquad (*)$$

where $h_{ij}, h_i \in \mathbf{Bool}$ and ρ_{ij} are regular formulas for all $1 \leq i, j \leq n$. Unguarded occurrences of variables X_j in the right-hand sides of the equations are obtained by taking $\rho_{ij} = \mathsf{nil}$. RESs of the form $(*)$ are encountered throughout the translation from a potentiality RES to its guarded form. For instance, a potentiality RES $\{X_1 \stackrel{\mu}{=} \mathsf{EF}_\rho Y\}$ can be rewritten as $\{X_1 \stackrel{\mu}{=} \mathsf{EF}_\rho X_2, X_2 \stackrel{\mu}{=} Y\}$, which is in the form above by considering $n = 2$, $h_{11} = \mathsf{false}$, $h_{12} = \mathsf{true}$, $\rho_{12} = \rho$, $h_1 = \mathsf{false}$, $h_{21} = h_{22} = \mathsf{false}$, and $h_2 = \mathsf{true}$. Similarly, a guarded potentiality RES is a particular case of form $(*)$ in which all regular formulas ρ_{ij} are simply atomic propositions p_{ij}.

To each propositional variable X_i of the potentiality RES $(*)$ and environment δ is associated a *path predicate* $P_{\delta,i} : Path_K \to \mathbf{Bool}$ characterizing the paths denoted by X_i in the context of δ. These path predicates are defined by the following equation system:

$$\left\{ P_{\delta,i}(\pi) \stackrel{\mu}{=} \bigvee_{j=1}^{n} \big(h_{ij} \wedge \exists l_{ij} \geq 0.\pi_{0,l_{ij}} \models \rho_{ij} \wedge P_{\delta,j}(\pi_{l_{ij},\infty})\big) \vee (h_i \wedge \pi_0 \in \delta(Y)) \right\}_{1\leq i\leq n}$$

where $\pi_{l_{ij},\infty}$ denotes the suffix of path π starting at the state of index l_{ij}.

The translation from a potentiality MES $\{X_1 \stackrel{\mu}{=} \mathsf{EF}_\rho Y\}$ to its guarded form preserves the path predicate associated to the main variable X_1, as shown by the lemma below.

Lemma 6 *Let K be a Kripke structure, $R = \{X_1 \stackrel{\mu}{=} \mathsf{EF}_\rho Y\}$ be an equation block, M be its corresponding guarded potentiality* MES *in the form $(*)$, and $P_{\delta,i}$ be its associated path predicates. Then:*

$$P_{\delta,1}(\pi) = \exists l \geq 0.\pi_{0,l} \models \rho \wedge \pi_l \in \delta(Y)$$

for any $\pi \in Path_K$ and any propositional environment δ.

Proof Let K be a Kripke structure and δ be a propositional environment. Let the equation block $R = \{X_1 \stackrel{\mu}{=} \mathsf{EF}_\rho X_2, X_2 \stackrel{\mu}{=} Y\}$ in form $(*)$. Its associated path predicates are defined as follows:

$$P_{\delta,1}(\pi) = \exists l_{12} \geq 0.\pi_{0,l_{12}} \models \rho_{12} \wedge P_{\delta,2}(\pi_{l_{12},\infty})$$
$$P_{\delta,2}(\pi) = \pi_0 \in \delta(Y)$$

where $\pi \in Path_K$. After appropriate renamings, we obtain the desired equality:

$$P_{\delta,1}(\pi) = \exists l \geq 0.\pi_{0,l} \models \rho \wedge \pi_l \in \delta(Y).$$

It remains to show that this equality also holds along the translation of R into guarded form. This translation consists of two phases: elimination of the regular operators present in ρ (Proposition 2) and elimination of unguarded occurrences of variables (Lemma 1). The substitutions performed in both phases preserve the path predicates associated to the variables defined by the substituted equations. This can be shown using similar arguments as in Proposition 2 and Lemma 1; we show below the path predicate preservation only for the first rule in Proposition 2, leaving the other ones as exercises for the interested reader.

This rule transforms the RES $R = \{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1.\rho_2} Y\}$ into the RES $R' = \{X \stackrel{\mu}{=} \mathsf{EF}_{\rho_1} Z, Z \stackrel{\mu}{=} \mathsf{EF}_{\rho_2} Y\}$. The predicate $P'_{\delta,1}$ associated to X_1 in R' is defined as follows:

$$\begin{aligned}
P'_{\delta,1}(\pi) &= \exists l \geq 0.\pi_{0,l} \models \rho_1 \wedge P'_{\delta,2}(\pi_{l,\infty}) && \text{by def. of } P'_{\delta,1} \\
&= \exists l \geq 0.\pi_{0,l} \models \rho_1 \wedge \exists l' \geq 0.\pi_{l,l+l'} \models \rho_2 \wedge \pi_{l+l'} \in \delta(Y) && \text{by def. of } P'_{\delta,2} \\
&= \exists l \geq 0.\exists l' \geq 0.\pi_{0,l} \models \rho_1 \wedge \pi_{l,l+l'} \models \rho_2 \wedge \pi_{l+l'} \in \delta(Y) \\
&= \exists k \geq 0.\exists j \geq 0.\pi_{0,j} \models \rho_1 \wedge \pi_{j,k} \models \rho_2 \wedge \pi_k \in \delta(Y) \\
&\qquad \text{by taking } k = l + l' \text{ and } j = l \\
&= \exists k \geq 0.\exists j \geq 0.\pi_{0,j} \models \rho_1 \wedge \pi_{j,k} \models \rho_2 \wedge \pi_k \in \delta(Y) && \text{by def. of } \rho_1.\rho_2
\end{aligned}$$

which coincides with the definition of $P_{\delta,1}$ in R. Thus, the path predicate $P_{\delta,1}$ associated to X_1 in R remains unchanged during the translation of R into guarded form, which shows the desired equality. □

The relation between the path predicates associated to a guarded potentiality MES and the interpretation of the corresponding determinized MES is given by the lemma below.

Lemma 7 *Let K be a Kripke structure, M be a guarded potentiality* Mes *in the form $(*)$, and $P_{\delta,i}$ be its associated path predicates. The determinized* Mes *corresponding to M is defined as in Section 3.2.3.2. Then:*

$$\Big(\Big[\!\Big[\Big\{X_I \stackrel{\mu}{=} \bigvee_{\emptyset \subset Q \subseteq prop(I)} \mathsf{AF}_Q X_{vars(Q,I)} \vee (h(I) \wedge Y)\Big\}_{I \subseteq [1,n]}\Big]\!\Big]_K \delta\Big)(X_J)$$
$$=$$
$$\{s \in S \mid \forall \pi \in Path_K(s).\exists j \in J.P_{\delta,j}(\pi)\}$$

for any index set $J \subseteq [1,n]$ and any propositional environment δ.

Proof Let K, M, δ, and $P_{\delta,i}$ as stated in the hypothesis. The functional $\Phi_\delta : (2^S)^{2^n-1} \to (2^S)^{2^n-1}$ associated to the determinized Mes corresponding to M is defined as follows:

$$\Phi_\delta(\langle U_J \rangle_{J \subseteq [1,n]}) = \Big\langle \Big[\!\Big[\bigvee_{\emptyset \subset Q \subseteq prop(I)} \mathsf{AF}_Q X_{vars(Q,I)} \vee (h(I) \wedge Y)\Big]\!\Big]_K (\delta \oslash [U_J/X_J]_{J \subseteq [1,n]})\Big\rangle_{I \subseteq [1,n]}$$

The interpretation of the determinized Mes is equal to $\mu\Phi_\delta$. Let $U = \langle \{s \in S \mid \forall \pi \in Path_K(s).\exists j \in J.P_{\delta,j}(\pi)\}\rangle_{J \subseteq [1,n]}$. We must show that $\mu\Phi_\delta = U$, which we split into a double inclusion.

Inclusion $\subseteq$. By Tarski's theorem [128], showing that $\mu\Phi_\delta \subseteq U$ amounts to show that $\Phi_\delta(U) \subseteq U$. We have:

$$\Phi_\delta(U) = \Big\langle \Big[\!\Big[\bigvee_{\emptyset \subset Q \subseteq prop(I)} \mathsf{AF}_Q X_{vars(Q,I)} \vee (h(I) \wedge Y)\Big]\!\Big]_K (\delta \oslash [\{s \in S \mid \forall \pi \in Path_K(s).\exists j \in J.P_{\delta,j}(\pi)\}/X_J]_{J \subseteq [1,n]})\Big\rangle_{I \subseteq [1,n]}$$

Let $I \subseteq [1,n]$ and $s \in (\Phi_\delta(U))_I$. By using the definition of Φ_δ and the interpretation of AF, and doing a simple first order reasoning, this is equivalent to the disjunction of the two conditions below:

(a) $\exists \emptyset \subset Q \subseteq prop(I).(s \models Q \wedge \forall \pi \in Path_K(s).\exists j \in vars(Q,I).P_{\delta,j}(\pi_{1,\infty}))$
(b) $h(I) \wedge s \in \delta(Y)$.

We must show that $s \in U_I$, i.e., that $\forall \pi \in Path(s).\exists i \in I.P_{\delta,i}(\pi)$. By applying the definition of path predicates, this expands as follows:

$$\forall \pi \in Path(s).\exists i \in I.(\exists j \in [1,n].(h_{ij} \wedge s \models p_{ij} \wedge P_{\delta,j}(\pi_{1,\infty})) \vee (h_i \wedge s \in \delta(Y)))$$

which is equivalent to the disjunction of the two conditions below:

(a') $\forall \pi \in Path(s).\exists j \in [1,n].(\exists i \in I.(h_{ij} \wedge s \models p_{ij}) \wedge P_{\delta,j}(\pi_{1,\infty}))$
(b') $\exists i \in I.h_i \wedge s \in \delta(Y)$.

Two cases are possible, depending on the fact that (a) or (b) holds.

Case (a). Let $Q \subseteq prop(I)$ such that $s \in Q$ and for all $\pi \in Path_K(s)$ there exists $j \in vars(Q,I)$ such that $P_{\delta,j}(\pi_{1,\infty})$. Let $\pi \in Path_K(s)$. From condition (a), we can choose $j \in vars(Q,I)$ such that $P_{\delta,j}(\pi_{1,\infty})$. Based on the definition of $vars(Q,I)$, we can choose $i \in I$ such that $p_{ij} \in Q$ and $h_{ij} = \mathsf{true}$. Since $s \models Q$ and $p_{ij} \in Q$, it follows that $s \models p_{ij}$ (recall from Section 3.2.3.2 that Q stands for the conjunction of all atomic propositions that it contains). This implies condition (a').

Case (b). Assume that $h(I) = \mathsf{true}$ and $s \in \delta(Y)$. From the definition of $h(I)$, it follows that we can choose $i \in I$ such that $h_i = \mathsf{true}$. This implies condition (b').

Inclusion $\supseteq$. The equation system defining the path predicates associated to M is defined as follows:

$$\left\{ P_{\delta,j}(\pi) \stackrel{\mu}{=} \bigvee_{k=1}^{n} (h_{jk} \wedge \pi_0 \models p_{jk} \wedge P_{\delta,k}(\pi_{1,\infty})) \vee (h_j \wedge \pi_0 \in \delta(Y)) \right\}_{1 \leq j \leq n}$$

For simplicity, we assume that all predicates occurring in the right-hand sides of equations are defined by some other equations of the system; this corresponds to the fact that M does not have free propositional variables excepting Y, whose interpretation is given by the environment δ. The functional $\Pi_\delta : (Path_K \to \mathbf{Bool})^n \to (Path_K \to \mathbf{Bool})^n$ associated to this system is defined below:

$$\Pi_\delta(P_1, ..., P_n) = \left\langle \lambda\pi. \left(\bigvee_{k=1}^{n} (h_{jk} \wedge \pi_0 \models p_{jk} \wedge P_k(\pi_{1,\infty})) \vee (h_j \wedge \pi_0 \in \delta(Y)) \right) \right\rangle_{1 \leq j \leq n}$$

It is straightforward to check that the functional Π_δ is continuous on the lattice $\langle (Path_K \to \mathbf{Bool})^n, (\lambda\pi.\mathsf{false})^n, (\lambda\pi.\mathsf{true})^n, \sqcap, \sqcup \rangle$, where $\sqcup$ and $\sqcap$ are the pointwise extensions of disjunction and conjunction operations on path predicates. Therefore, its minimal fixed point $\mu\Pi_\delta$, which gives the interpretation of the equation system, has the following iterative characterization [80]:

$$\mu\Pi_\delta = \bigsqcup_{k \geq 0} \Pi_\delta^k((\lambda\pi.\mathsf{false})^n), \qquad \Pi_\delta^0((\lambda\pi.\mathsf{false})^n) = (\lambda\pi.\mathsf{false})^n.$$

We note $\langle P_{\delta,j}^k \rangle_{1 \leq j \leq n} = \Pi_\delta^k((\lambda\pi.\mathsf{false})^n)$. From the iterative characterization of $\mu\Pi_\delta$ and

the definition of $\sqcup$, we have:

$$P_{\delta,j}(\pi) = \left(\bigsqcup_{k \geq 0} \langle P^k_{\delta,j} \rangle_{1 \leq j \leq n} \right)(\pi) = \exists k \geq 0 . P^k_{\delta,j}(\pi).$$

To obtain the desired inclusion, we use the following statement:

$$\forall k \geq 0. \left(\left\langle \{ s \in S \mid \forall \pi \in Path_K(s). \exists j \in J. P^k_{\delta,j}(\pi) \} \right\rangle_{J \subseteq [1,n]} \subseteq \mu\Phi_\delta \right) \qquad (**)$$

To show that $U \subseteq \mu\Phi_\delta$, let $J \subseteq [1,n]$ and let $s \in U_J$, which means that for all $\pi \in Path_K(s)$, there exists $j \in J$ such that $P_{\delta,j}(\pi)$. The definition of $P_{\delta,j}(\pi)$ above ensures that we can find $k \geq 0$ such that $P^k_{\delta,j}(\pi)$. By applying $(**)$ for that k, we obtain $s \in (\mu\Phi_\delta)_J$, which implies in turn the desired inclusion $U \subseteq \mu\Phi_\delta$.

It remains to show the $(**)$ statement. We proceed by induction on k.

Base step.

$\left\langle \{ s \in S \mid \forall \pi \in Path_K(s). \exists j \in J. P^0_{\delta,j}(\pi) \} \right\rangle_{J \subseteq [1,n]} =$ by def. $\Pi^0_\delta((\lambda\pi.\mathsf{false})^n)$

$\langle \{ s \in S \mid \forall \pi \in Path_K(s). \exists j \in J. \mathsf{false} \} \rangle_{J \subseteq [1,n]} = \langle \emptyset \rangle_{J \subseteq [1,n]} \subseteq \mu\Phi_\delta.$

Inductive step. Let $U^k = \langle \{ s \in S \mid \forall \pi \in Path_K(s). \exists j \in J. P^k_{\delta,j}(\pi) \} \rangle_{J \subseteq [1,n]}$. We show below that $U^{k+1} \subseteq \Phi_\delta(U^k)$, which together with the inductive hypothesis and the definition of minimal fixed points implies $U^{k+1} \subseteq \Phi_\delta(U^k) \subseteq \Phi_\delta(\mu\Phi_\delta) = \mu\Phi_\delta$, i.e., the desired inequality.

Let $J \subseteq [1,n]$ and let $s \in (U^{k+1})_J$, which means that for every $\pi \in Path_K(s)$ there exists $j \in J$ such that $P^{k+1}_{\delta,j}(\pi)$. From the definition of Π_δ and $P^k_{\delta,j}$, we have:

$$P^{k+1}_{\delta,j}(\pi) = \bigvee_{l=1}^{n} (h_{jl} \wedge \pi_0 \models p_{jl} \wedge P^k_{\delta,l}(\pi_{1,\infty})) \vee (h_j \wedge \pi_0 \in \delta(Y)).$$

By expanding this equality and doing a simple first order reasoning, the conditions above can be rewritten as the disjunction of the two conditions below:

(c) $\forall \pi \in Path_K(s). \exists l \in [1,n]. (\exists j \in J. (h_{jl} \wedge s \models p_{jl}) \wedge P^k_{\delta,l}(\pi_{1,\infty}))$
(d) $\exists j \in J. h_j \wedge s \in \delta(Y)$.

Let $s \in (\Phi_\delta(U^k))_J$. From the definition of Φ_δ, this is equivalent to:

$$s \in \left[\!\!\left[\bigvee_{\emptyset \subset Q \subseteq prop(J)} \mathsf{AF}_Q X_{vars(Q,J)} \vee (h(J) \wedge Y) \right]\!\!\right]_K (\delta \oslash [(U^k)_L / X_L]_{L \subseteq [1,n]})$$

Using the definition of U^k and the interpretation of AF, and doing a simple first order reasoning, this is equivalent to the disjunction of the two conditions below:

(c') $\exists \emptyset \subset Q \subseteq prop(J).(s \models Q \wedge \forall \pi \in Path_K(s).\exists l \in vars(Q,J).P^k_{\delta,l}(\pi_{1,\infty}))$
(d') $h(J) \wedge s \in \delta(Y)$.

Two cases are possible, depending on the fact that (c) or (d) holds.

Case (c). Let the set of atomic propositions Q be defined as follows:

$$Q = \bigcup_{\pi \in Path_K(s)} \{p_{jl} \mid j \in J \wedge l \in [1,n] \wedge s \models p_{jl}\}$$

Condition (c) guarantees that Q is not empty and the definition of $prop(J)$ implies that $Q \subseteq prop(J)$. Since $s \models p_{jl}$ for every $p_{jl} \in Q$, it follows that $s \models Q$ (recall from Section 3.2.3.2 that Q stands for the conjunction of all atomic propositions that it contains). Let $\pi \in Path_K(s)$. From condition (c), we can find $l \in [1,n]$ and $j \in J$ such that h_{jl} and $s \models p_{jl}$ and $P_{\delta,j}(\pi_{1,\infty})$. Since $p_{jl} \in Q$ by definition of Q, from the definition of $vars(Q,J)$ it follows that $l \in vars(Q,J)$. This implies condition (c').

Case (d). Let $j \in J$ such that $h_j = \mathsf{true}$. From the definition of $h(J)$, it follows that $h(J) = \mathsf{true}$. Since $s \in \delta(Y)$ from condition (d), this implies condition (d').

This concludes the proof of the lemma. □

We are finally ready to prove Proposition 3.

Proof *(Proposition 3).*

Let K be a Kripke structure, δ be a propositional environment, $R = \{X_1 \stackrel{\mu}{=} \mathsf{AF}_\rho Y\}$ an equation block. Let $P_{\delta,i}$ be the path predicates associated to the guarded potentiality MES obtained by translating R, and let M be the MES further obtained after determinization.

We have:

$(\llbracket M \rrbracket_K \delta)(X_{\{1\}}) =$	by Lemma 7
$\{s \in S \mid \forall \pi \in Path_K(s).P_{\delta,1}(\pi)\} =$	by Lemma 6
$\{s \in S \mid \forall \pi \in Path_K(s).\exists l \geq 0.\pi_{0,l} \models \rho \wedge \pi_l \in \delta(Y)\} =$	by def. of $\mathsf{AF}_\rho Y$ and $\llbracket\ \rrbracket$
$(\llbracket \{X_1 \stackrel{\mu}{=} \mathsf{AF}_\rho Y\} \rrbracket_K \delta)(X_1)$.	

□

Appendix B

GNA modeling and simulation tool screenshots

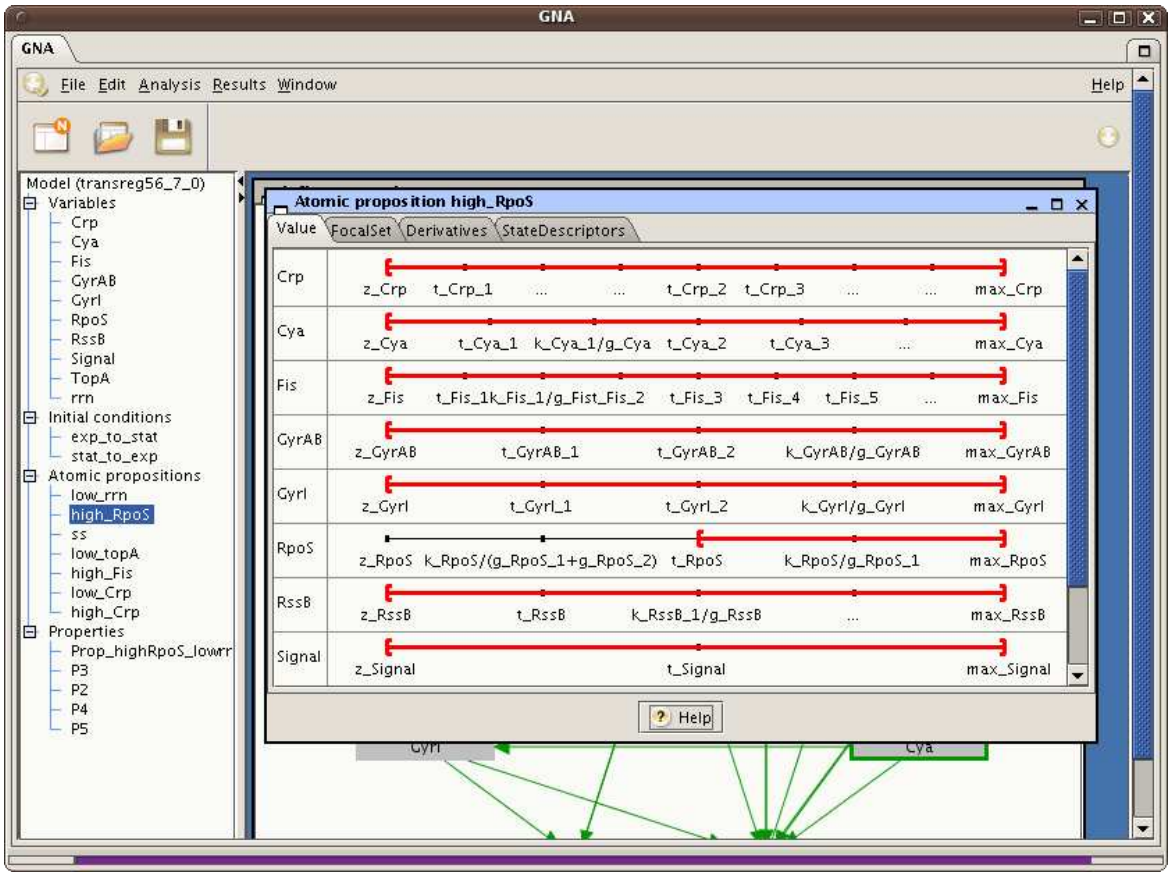


Figure B.1: Atomic proposition specification window, where atomic propositions are defined in terms of restrictions applied to a state (e.g., restrictions on concentration values, focal sets, derivatives, and other state descriptors). In this case, the value of the concentration is restricted to lie below the threshold *high_RpoS*.

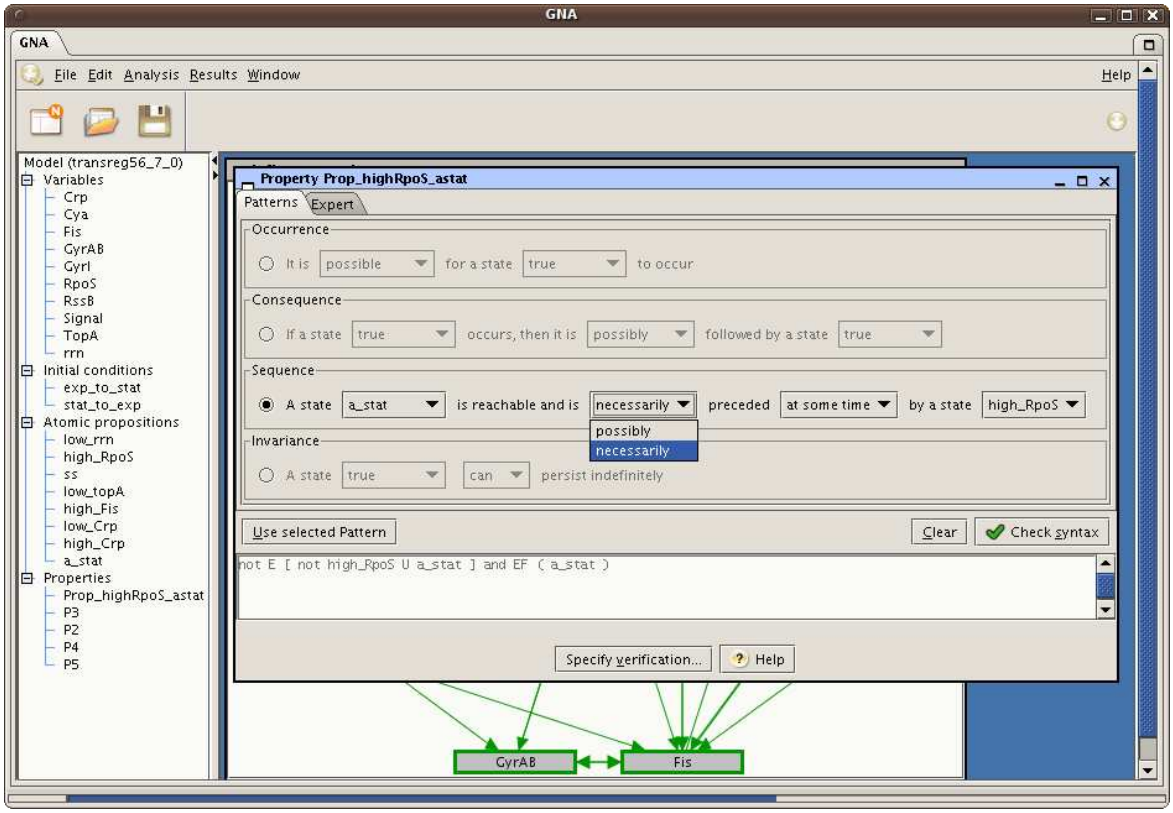


Figure B.2: Graphical user interface for the specification of biological properties. The modeler can use a pattern-based property editor for frequently-asked questions, and a text editor for the specification of more complex biological properties (expert mode). In this case, the sequence pattern is selected and instantiated with the a_{stat} and the *high_RpoS* atomic propositions.

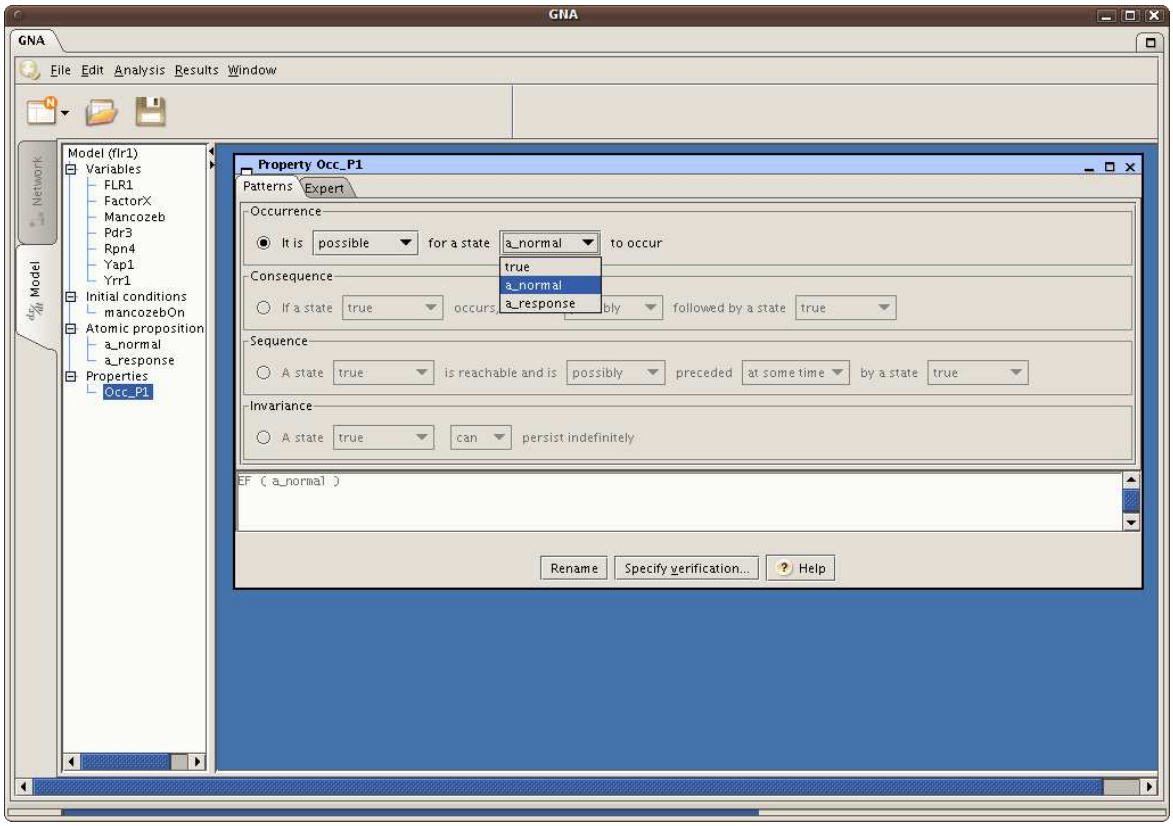


Figure B.3: Graphical user interface for the specification of biological properties. The modeler can use a pattern-based property editor for frequently-asked questions, and a text editor for the specification of more complex biological properties (expert mode). In this case, the occurrence pattern is selected and instantiated with the a_{normal} atomic proposition.

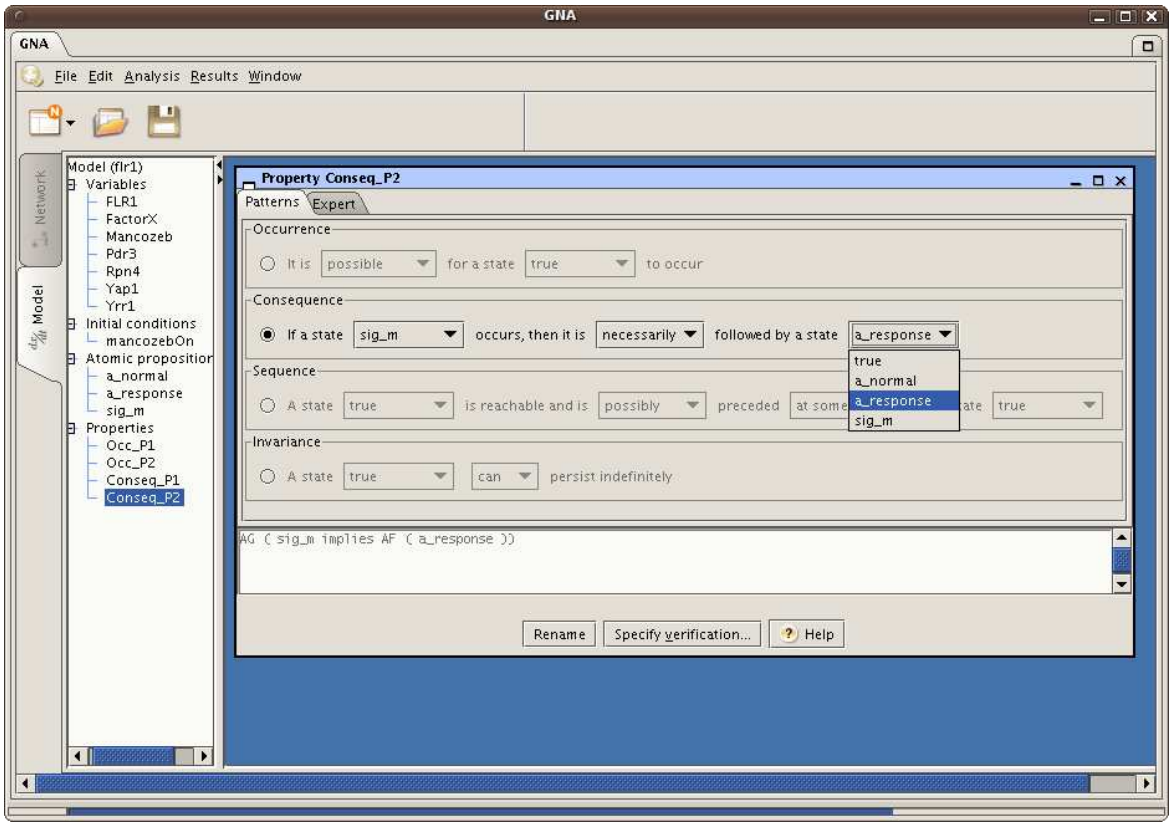


Figure B.4: Graphical user interface for the specification of biological properties. The modeler can use a pattern-based property editor for frequently-asked questions, and a text editor for the specification of more complex biological properties (expert mode). In this case, the consequence pattern is selected and instantiated with the sig_m and $a_{response}$ atomic propositions.

Bibliography

[1] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986. 46

[2] A.M. Alarco, I. Balan, D. Talibi, N. Mainville, and M. Raymond. Ap1-mediated multidrug resistance in *Saccharomyces cerevisiae* requires FLR1 encoding a transporter of the major facilitator superfamily. *J. Biol. Chem.*, 272:19304–19313, 1997. 92

[3] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*. Oxford University Press, New York, NY, 1977. 54

[4] M. Antoniotti, A. Policriti, N. Ugel, and B. Mishra. Model building and model checking for biochemical processes. *Cell Bioch. Biophys.*, 38(3):271–286, 2003. 4, 21, 51, 65

[5] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification language. In Joost-Pieter Katoen and Perdita Stevens, editors, *Proc. 8th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *LNCS*, pages 296–211. Springer Verlag, April 2002. 50

[6] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous time markov chains. *ACM T. Comput. Log.*, 1:162–170, 2002. 23

[7] V. Baldazzi, D. Ropers, Y. Markowicz, D. Kahn, J. Geiselmann, and H. de Jong. The carbon assimilation network in *E. coli* is densely connected and largely sign-determined by directions of metabolic fluxes. *PLoS Comput. Biol.*, under revision, 2010. 102

[8] P. Ballarini, T. Mazza, A. Palmisano, and A. Csikasz-Nagy. Studying irreversible transitions in a model of cell cycle regulation. In *Proc. 3rd Intl. Workshop on Practical Applications of Stochastic Modelling, (PASM 2008)*, 2009. 4, 21, 65

[9] J. Barnat, L. Brim, I. Cerná, S. Drazan, and D. Safranek. Parallel model checking large-scale genetic regulatory networks with DiVinE. In *Proc. 1st Intl. Workshop From*

Biology To Concurrency and Back (FBTC 2007), volume 194 of *ENTCS*, 2008. 4, 21, 51, 65

[10] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In Bernhard Steffen and Giorgio Levi, editors, *Proc. 5th Intl. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI 2004)*, volume 2937 of *LNCS*, pages 44–57. Springer Verlag, January 2004. 50

[11] G. Batt. *Validation de modèles qualitatifs de réseaux de régulation génique : une méthode basée sur des techniques de vérification formelle*. PhD thesis, Université Joseph Fourier - Grenoble I, 2006. 56

[12] G. Batt, D. Bergamini, H. de Jong, H. Gavarel, and R. Mateescu. Model checking genetic regulatory networks using GNA and CADP. In S. Graf and L. Mounier, editors, *Proc. 11th Intl. Workshop on Model Checking of Software (SPIN 2004)*, volume 2989 of *LNCS*, pages 158–163, Berlin, 2004. Springer Verlag. 68, 77

[13] G. Batt, H. de Jong, M. Page, and J. Geiselmann. Symbolic reachability analysis of genetic regulatory networks using discrete abstractions. *Automatica*, 44(4):982–989, 2008. 66, 67

[14] G. Batt, D. Ropers, H. de Jong, J. Geiselmann, R. Mateescu, M. Page, and D. Schneider. Validation of qualitative models of genetic regulatory networks by model checking: Analysis of the nutritional stress response in *Escherichia coli*. *Bioinformatics*, 21(Suppl 1):i19–i28, 2005. 4, 21, 29, 47, 51, 56, 64, 65, 66, 68, 77

[15] G. Batt, B. Yordanov, C. Belta, and R. Weiss. Robustness analysis and tuning of synthetic gene networks. *Bioinformatics*, 23(18):2415–2422, 2007. 4, 21, 65, 75

[16] I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic Sugar. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proc. 13th Intl. Conf. on Computer Aided Verification (CAV 2001)*, volume 2102 of *LNCS*, pages 363–367. Springer Verlag, July 2001. 35, 50

[17] I. Beer, S. Ben-David, and A. Landver. On-the-fly model checking of RCTL formulas. In Alan Hu and Moshe Y. Vardi, editors, *Proc. 10th Intl. Conf. Computer Aided Verification (CAV 1998)*, volume 1427 of *LNCS*, pages 184–194. Springer Verlag, June 1998. 29, 34, 35, 36, 50

[18] G. Bernot, J.-P. Comet, A. Richard, and J. Guespin. Application of formal methods to biological regulatory networks: Extending Thomas' asynchronous logical approach with temporal logic. *J. Theor. Biol.*, 229(3):339–348, 2004. xi, 4, 21, 22, 51, 56, 65, 77

[19] K. Bettenbrock, S. Fischer, A. Kremling, K. Jahreis, T. Sauter, and E.-D. Gilles. A quantitative approach to catabolite repression in *Escherichia coli*. *J. Biol. Chem.*, 281(5):2578–2584, 2006. 82

[20] H. Bolouri and E.H. Davidson. The gene regulatory network basis of the "community effect," and analysis of a sea urchin embryo example. *Dev. Biol.*, in press, 2009. 21, 75

[21] D. Bosnacki, H.M.M. ten Eikelder, M.N. Steijaert, and E.P. de Vink. Stochastic analysis of amino acid substitution in protein synthesis. In M. Heiner and A.M. Uhrmacher, editors, *Proc. 6th Conf. Computational Methods in Systems Biology (CMSB 2008)*, volume 5307 of *LNCS*, pages 367–386. Springer Verlag, Berlin, 2008. 4, 21, 65

[22] T. Brázdil and I. Cerná. Model checking of RegCTL. *Comput. Artif. Intell.*, 25(1), 2006. 29, 34, 35, 36, 50

[23] N. Brôco, S. Tenreiro, C.A. Viegas, and I. Sá-Correia. FLR1 gene (ORF YBR008c) is required for benomyl and methotrexate resistance in *Saccharomyces cerevisiae* and its benomyl-induced expression is dependent on Pdr3 transcriptional regulator. *Yeast*, 15:1595–1608, 1999. 92

[24] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE T. Comput.*, 35:677–691, 1986. 18

[25] J.A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964. 43

[26] M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton. Analysis of signalling pathways using the PRISM model checker. In G. Plotkin, editor, *Proc. 3th Conf. Computational Methods in Systems Biology (CMSB 2005)*, pages 79–90, Edinburgh, Scotland, 2005. 4, 21, 22, 24, 51, 56, 65, 76

[27] L. Calzone, F. Fages, and S. Soliman. BIOCHAM: An environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807, 2006. 26, 75, 77

[28] N. Chabrier and F. Fages. Symbolic model checking of biochemical networks. In *Proc. 1st Intl. Workshop Computational Methods in Systems Biology (CMSB 2003)*, volume 2602 of *LNCS*, pages 149–162, Rivereto, Italy, 2003. Springer-Verlag. 24

[29] N. Chabrier-Rivier, M. Chiaverini, V. Danos, F. Fages, and V. Schächter. Modeling and querying biomolecular interaction networks. *Theor. Comput. Sci.*, 325(1):25–44, 2004. 4, 21, 49, 51, 55, 56, 64, 65, 77

[30] C. Chaouiya. Petri net modelling of biological networks. *Brief. Bioinform.*, 8(4):210–219, 2007. 51

[31] C. Chaouiya, E. Remy, and D. Thieffry. Qualitative petri net modelling of genetic networks. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Transactions on Computational Systems Biology VI*, volume 4220 of *LNCS*, pages 95–112, Berlin, 2006. Springer-Verlag. 21, 75

[32] C. Chassagnole, N. Noisommit-Rizzi, J.W. Schmid, K. Mauch, and M. Reuss. Dynamic modeling of the central carbon metabolism of *Escherichia coli*. *Biotechnol. Bioeng.*, 79(1):53–73, 2002. 82

[33] K.C. Chen, L. Calzone, A. Csikasz-Nagy, F.R. Cross, B. Novak, and J.J. Tyson. Integrative analysis of cell cycle control in budding yeast. *Mol. Biol. Cell*, 15(8):3841–3862, 2004. 3, 29

[34] W.W. Chen, B. Schoeberl, P.J. Jasper, M. Niepel, U.B. Nielsen, D.A. Lauffenburger, and P.K. Sorger. Input-output behavior of ErbB signaling pathways as revealed by a mass action model trained against dynamic data. *Mol. Syst. Biol.*, 5(239), 2009. 3

[35] K.H. Cho, S.Y. Shin, H.W. Kim, O. Wolkenhauer, B. McFerran, and W. Kolch. Mathematical modeling of the influence of rkip on the erk signaling pathway. In *Proc. 1st Intl. Workshop Computational Methods in Systems Biology (CMSB 2003)*, volume 2602 of *LNCS*, pages 127–141. Springer-Verlag, 2003. xi, 22, 23, 76

[36] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV2: An OpenSource tool for symbolic model checking. In D. Brinksma and K.G. Larsen, editors, *Proc. 14th Intl. Conf. on Computer Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, pages 359–364. Springer Verlag, Berlin, 2002. 55, 75, 76

[37] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *Springer Intl. J. Software Tool. Tech. Tran.*, 2(4):410–425, apr 2000. 24

[38] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics, school/workshop. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 428–437. Springer-Verlag, 1988. 14

[39] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8:244–263, 1986. 5, 11, 18, 25, 34

[40] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000. xi, xv, 3, 4, 10, 11, 13, 15, 30, 35, 67, 68

[41] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *Form. Method. Syst. Des.*, 2(2):121–147, April 1993. 48

[42] S. Le Crom, F. Devaux, P. Marc, X. Zhang, W.S. Moye-Rowley, and C. Jacq. New insights into the pleiotropic drug resistance network from genome-wide characterization of the yrr1 transcription factor regulation system. *Mol. Cell. Biol.*, 22(8):2642–2649, 2002. 95

[43] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theor. Comput. Sci.*, 126(1):77–96, April 1994. 16, 35

[44] H. de Jong. Modeling and simulation of genetic regulatory systems : A literature review. *J. Comput. Bio.*, 9(1):69–105, 2002. 21, 75

[45] H. de Jong, J. Geiselmann, C. Hernandez, and M. Page. Genetic Network Analyzer: Qualitative simulation of genetic regulatory networks. *Bioinformatics*, 19(3):336–344, 2003. 26, 66, 75

[46] H. de Jong and M. Page. Search for steady states of piecewise-linear differential equation models of genetic regulatory networks. *ACM/IEEE Trans. Comput. Biol. Bioinform.*, 5(2):508–522, 2008. 93

[47] J. DeRisi, B. van den Hazel, P. Marc, E. Balzi, P. Brown, C. Jacq, and A. Goffeau. Genome microarray analysis of transcriptional activation in multidrug resistance yeast mutants. *FEBS Letters*, 470:156–160, 2000. 91

[48] P.J. Dias, M.C. Teixeira, J.P. Telo, and I. Sá-Correia. Insights into the mechanisms of toxicity and tolerance to the agricultural fungicide mancozeb in yeast, as suggested by a chemogenomic approach. *OMICS: A Journal of Integrative Biology*, 14:in press, 2010. 92

[49] D. Dubnau and R. Losick. Bistability in bacteria. *Mol. Microbiol.*, 61(3):564–572, 2006. 29, 39

[50] M.B. Dwyer, G.S. Avrunin, and J.C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. 21st Intl. Conf. Software Engineering (ICSE 1999)*, pages 411–420, Los Alamitos, CA, 1999. xii, 53, 54, 63, 64, 100

[51] E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time. In *Proc. 10th Annual ACM Symposium on Principles of Programming Languages (POPL 1983)*, pages 127–140, 1983. Also appeared in Journal of ACM, 33(1):151-178, 1986. 15, 25, 29, 49

[52] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proc. 1st Intl. Symposium on Logic in Computer Science (LICS 1986)*, pages 267–278, 1986. 35, 47

[53] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Sci. Comput. Program.*, 8:275–306, 1987. 34

[54] T. Erl. *Service-oriented architecture*. Prentice Hall PTR, New Jersey, 2004. 65

[55] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, September 1979. 29

[56] J. Fisher and T.A. Henzinger. Executable cell biology. *Nat. Biotechnol.*, 25(11):1239–1250, 2007. 3, 51

[57] J. Fisher, N. Piterman, A. Hajnal, and T.A. Henzinger. Predictive modeling of signaling crosstalk during *C. elegans* vulval development. *PLoS Comput. Biol.*, 3(5):e92, 2007. 4, 21, 51, 65

[58] D.B. Forger and C.S. Peskin. A detailed predictive model of the mammalian circadian clock. *Proc. Natl. Acad. Sci. USA*, 100(25):14806–14811, 2003. 3

[59] E. Gamma, R. Helm, R. Johnson, and J.M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, New York, NY, 1994. 54

[60] H. Garavel. OPEN/CÆSAR: An open software architecture for verification, simulation, and testing. In Bernhard Steffen, editor, *Proc. 1st Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1998)*, volume 1384 of *LNCS*, pages 68–84, Berlin, March 1998. Springer Verlag. Full version available as INRIA Research Report RR-3352. 48

[61] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2006: A toolbox for the construction and analysis of distributed processes. In Werner Damm and Holger Hermanns, editors, *Proc. 19th Intl. Conf. on Computer Aided Verification (CAV 2007)*, volume 4590 of *LNCS*, pages 158–163. Springer Verlag, July 2007. 6, 7, 30, 35, 47

[62] H. Garavel, R. Mateescu, F. Lang, and W. Serwe. CADP 2006: A toolbox for the construction and analysis of distributed processes. In W. Damm and H. Hermanns, editors, *Proc. 19th Intl. Conf. Computer Aided Verification (CAV 2007)*, volume 4590 of *LNCS*, pages 158–163, Berlin, 2007. Springer-Verlag. 75

[63] T.S. Gardner, C.R. Cantor, and J.J. Collins. Construction of a genetic toggle switch in *Escherichia coli*. *Nature*, 403(6767):339–342, 2000. 85

[64] D. Gilbert, M. Heiner, and S. Lehrack. A unifying framework for modelling and analysing biochemical pathways using petri nets. In M. Calder and S. Gilmore, editors, *Proc. 5th Conf. Computational Methods in Systems Biology (CMSB 2007)*, volume 4695 of *LNBI*, pages 200–216. Springer-Verlag, Berlin, 2007. 4, 21, 65, 75

[65] L. Glass and S.A. Kauffman. The logical analysis of continuous non-linear biochemical control networks. *J. Theor. Biol.*, 39(1):103–129, 1973. 66

[66] J.F. Guespin-Michel and M. Kaufman. Positive feedback circuits and adaptive regulations in bacteria. *Acta Biotheor.*, 49:207–218, 2001. 21, 56

[67] R.M. Gutierrez-Ríos, J.A. Freyre-Gonzalez, O. Resendis, J. Collado-Vides, M. Saier, and G. Gosset. Identification of regulatory network topological units coordinating the genome-wide transcriptional response to glucose in *Escherichia coli*. *BMC Microbiol.*, 7(1):53, 2007. 82

[68] K. Hamaguchi, H. Hiraishi, and S. Yajima. Branching time regular temporal logic for model checking with linear time complexity. In E. M. Clarke and R. P. Kurshan, editors, *Proc. 2nd Intl. Conf. on Computer Aided Verification (CAV 1990)*, volume 531 of *LNCS*, pages 253–262, Berlin, June 1990. Springer Verlag. 50

[69] T. Hardiman, K. Lemuth, M.A. Keller, M. Reuss, and M. Siemann-Herzberg. Topology of the global regulatory network of carbon limitation in *Escherichia coli*. *J. Biotechnol.*, 132(4):359–374, 2007. 82

[70] R. Hengge-Aronis. Regulation of gene expression during entry into stationary phase. In F.C. Neidhardt, R. Curtiss III, J.L. Ingraham, E.C.C. Lin, K.B. Low, B. Magasanik, W.S. Reznikoff, M. Riley, M. Schaechter, and H.E. Umbarger, editors, *Escherichia coli and Salmonella: Cellular and Molecular Biology*, pages 1497–1512, Washington DC, 1996. ASM Press. 87

[71] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32, 1985. 16

[72] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006)*, volume 3920 of *LNCS*, pages 441–444. Springer-Verlag, 2006. 56, 76

[73] G. Holzmann. *The SPIN Model Checker – Primer and Reference Manual*. Addison-Wesley, 2003. 35

[74] G.W. Huisman, D.A. Siegele, M.M. Zambrano, and R. Kolter. Morphological and physiological changes during stationary phase. In F.C. Neidhardt, editor, *Escherichia*

coli and Salmonella: Cellular and Molecular Biology, pages 1672–1682, Washington DC, 1996. ASM Press. 82

[75] IEEE. PSL: Property specification language. Standard P1850, IEEE Computer Society, September 2004. 50

[76] ISO/IEC. LOTOS — a formal description technique based on the temporal ordering of observational behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1989. 48

[77] N. Jamshidi and B.Ø. Palsson. Formulating genome-scale kinetic models in the post-genome era. *Mol. Syst. Biol.*, 4(171), 2008. 3

[78] W. Janssen, R. Mateescu, S. Mauw, P. Fennema, and P. van der Stappen. Model checking for managers. In Dennis Dams, Rob Gerth, Stefan Leue, and Mieke Massink, editors, *Proc. 6th Intl. Workshop on Model Checking of Software (SPIN 1999)*, volume 1680 of *LNCS*, pages 92–107, Berlin, 1999. Springer Verlag. 55

[79] H. Jungwirth and K. Kuchler. Yeast ABC transporters - a tale of sex, stress, drugs and aging. *FEBS Letters*, 580:1131–1138, 2006. 91

[80] S.C. Kleene. *Introduction to Metamathematics*. North-Holland, 1952. 109, 110, 112, 115, 117, 120

[81] E. Klipp, B. Nordlander, R. Krüger, P. Gennemark, and S. Hohmann. Integrative model of the response of yeast to osmotic shock. *Nat. Biotechnol.*, 23(8):975–982, 2005. xi, 3, 4

[82] I. Koch, B.H. Junker, and M. Heiner. Application of petri net theory for modelling and validation of the sucrose breakdown pathway in the potato tuber. *Bioinformatics*, 21(7):1219–1226, 2005. 21, 75

[83] K.W. Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Mol. Biol. Cell*, 10:2703–2734, 1999. 24, 55

[84] D. Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. 5, 15, 25, 29, 36, 99

[85] A. Kremling, S. Kremling, and K. Bettenbrock. Catabolite repression in *Escherichia coli* - a comparison of modelling approaches. *FEBS J.*, 276(2):594–602, 2009. 82

[86] S. Kripke. Semantical considerations in modal logic. *Acta Phil. Fennica*, 16:83–94, 1963. 10

[87] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000. xv, 15, 62

[88] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In J. Bradley T. Field, P. Harrison and U. Harder, editors, *Proc. TOOLS 2002*, volume 2324 of *LNCS*, pages 200–204. Springer-Verlag, 2002. 24

[89] L. Lamport. Proving the correctness of multiprocess programs. *IEEE T. Software Eng.*, 3:125–143, 1977. 54

[90] L. Lamport. "sometime" is sometimes "not never": on the temporal logic of programs. In *Proc. 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 174–185, New York, NY, USA, 1980. ACM. 15

[91] F. Lang. EXP.OPEN 2.0: A flexible tool integrating partial order, compositional, and on-the-fly verification methods. In Jaco van de Pol, Judi Romijn, and Graeme Smith, editors, *Proc. 5th Intl. Conf. on Integrated Formal Methods (IFM 2005)*, volume 3771 of *LNCS*. Springer Verlag, November 2005. Full version available as INRIA Research Report RR-5673. 48

[92] K.G. Larsen. Proof systems for Hennessy-Milner logic with recursion. In *Proc. 13th Colloquium on Trees in Algebra and Programming (CAAP 1988)*, volume 299 of *LNCS*, pages 215–230, Berlin, March 1988. Springer Verlag. 7, 17, 30, 35, 38

[93] T.I. Lee, N.J. Rinaldi, F. Robert, D.T. Odom, Z. Bar-Joseph, G.K. Gerber, N.M. Hannett, C.T. Harbison, C.M. Thompson, I. Simon, J. Zeitlinger, E.G. Jennings, H.L. Murray, D.B. Gordon, B. Ren, J.J. Wyrick, J.B. Tagne, T.L. Volkert, E. Fraenkel, D.K. Gifford, and R.A. Young. Transcriptional regulatory networks in *Saccharomyces cerevisiae*. *Science*, 298:799–804, 2002. 91

[94] J.-C. Leloup and A. Goldbeter. Toward a detailed computational model for the mammalian circadian clock. *Proc. Natl. Acad. Sci. USA*, 100(12):7051–7056, 2003. 3, 29

[95] F. Li, T. Long, Y. Lu, Q. Ouyang, and C. Tang. The yeast cell-cycle network is robustly designed. *Proc. Natl. Acad. Sci. USA*, 101(14):4781–4786, 2004. 21, 75

[96] A. Lucau-Danila, T. Delaveau, G. Lelandais, F. Devaux, and C. Jacq. Competitive promoter occupancy by two yeast paralogous transcription factors controlling the multidrug resistance phenomenon. *J. Biol. Chem.*, 278:52641–52650, 2003. 91

[97] A. Lucau-Danila, G. Lelandais, Z. Kozovska, V. Tanty, T. Delaveau, F. Devaux, and C. Jacq. Early expression of yeast genes affected by chemical stress. *Mol. Cell. Biol.*, 25(5):1860–1868, 2005. 91, 95

[98] Z. Manna and A. Pnueli. Tools and rules for the practicing verifier. In R. Rashid, editor, *Carnegie Mellon Computer Science: A 25th Anniversary Commemorative*, pages 125–159, New York, NY, 1991. ACM Press. 54, 63

[99] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, volume I: Specification*. Springer Verlag, 1992. 3, 25, 34, 60

[100] A.J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distrib. Comput.*, 1(4):226–234, 1986. 48

[101] R. Mateescu. CÆSAR_SOLVE: A generic library for on-the-fly resolution of alternation-free boolean equation systems. *Springer Intl. J. Software Tool. Tech. Tran.*, 8(1):37–56, February 2006. Full version available as INRIA Research Report RR-5948, July 2006. 48, 49

[102] R. Mateescu, P.T. Monteiro, E. Dumas, and H. de Jong. Computation tree regular logic for genetic regulatory networks. In Sung Deok Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *Proc. 6th Intl. Symposium on Automated Technology for Verification and Analysis (ATVA 2008)*, volume 5311 of *LNCS*, pages 48–63. Springer Verlag, oct 2008. Full version available as INRIA Research Report RR-6521. 6, 29, 49, 68

[103] R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Sci. Comput. Program.*, 46(3):255–281, March 2003. 30, 36, 48

[104] R. Mateescu and D. Thivolle. A model checking language for concurrent value-passing systems. In *Proc. 15th Intl. Symposium on Formal Methods (FM 2008)*, May 2008. 49, 50

[105] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993. 22

[106] P.T. Monteiro, P.J. Dias, D. Ropers, A.L. Oliveira, A.T. Freitas, I. Sá-Correia, and M.C. Teixeira. The regulatory network underlying the transcriptional up-regulation of the flr1 gene in mancozeb stressed yeast cells: qualitative modeling and simulation. In *Book of Abstracts of the Congresso Nacional MICRO-BIOTEC*, 2009. xiii, 6, 92

[107] P.T. Monteiro, P.J. Dias, D. Ropers, A.L. Oliveira, I. Sá-Correia, M.C. Teixeira, and A.T. Freitas. Qualitative modeling and formal verification of the flr1 gene mancozeb response in *Saccharomyces cerevisiae*. *BMC Systems Biology*, submitted, 2010. 6

[108] P.T. Monteiro, E. Dumas, B. Besson, R. Mateescu, M. Page, A.T. Freitas, and H. de Jong. A service-oriented architecture for integrating the modeling and formal verification of genetic regulatory networks. *BMC Bioinformatics*, 10:450, 2009. 6

[109] P.T. Monteiro, D. Ropers, R. Mateescu, A.T. Freitas, and H. de Jong. Temporal logic patterns for querying dynamic models of cellular interaction networks. *Bioinformatics*, 24(16):i227–i233, 2008. xiii, 6, 65, 77, 83

[110] A. Naldi, D. Berenguier, A. Fauré, F. Lopez, D. Thieffry, and C. Chaouiya. Logical modelling of regulatory networks with ginsim 2.3. *Biosystems*, 97(2):134–139, 2009. 26, 75, 77

[111] D.T. Nguyen, A.M. Alarco, and M. Raymond. Multiple Yap1p-binding sites mediate induction of the yeast major facilitator FLR1 gene in response to drugs, oxidants, and alkylating agents. *J. Biol. Chem.*, 276:1138–1145, 2001. 92

[112] T.J. Oh, I.L. Jung, and I.G. Kim. The *Escherichia coli* SOS gene *sbmC* is regulated by H-NS and RpoS during the SOS induction and stationary growth phase. *Biochem. Biophys. Res. Commun.*, 288(4):1052–1058, 2001. 89

[113] D. Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Proc. 5th GI-Conf. on Theoretical Computer Science*, volume 104 of *LNCS*, pages 167–183. Springer-Verlag, 1981. 10

[114] H. Qi, R. Menzel, and Y.C. Tse-Dinh. Regulation of *Escherichia coli topA* gene transcription: involvement of a *sigmaS*-dependent promoter. *J. Mol. Biol.*, 267(3):481–489, 1997. 88

[115] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Proc. 5th Colloquium on International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351, Berlin, 1982. Springer Verlag. 18

[116] A. Rizk, G. Batt, F. Fages, and S. Soliman. A general computational method for robustness analysis with applications to synthetic gene networks. *Bioinformatics*, 25:i169–i178, 2009. 24

[117] J. Rohwer, N. Meadow, S. Roseman, H. Westerhoff, and P. Postma. Understanding glucose transport by the bacterial phosphoenolpyruvate:glycose phosphotransferase system on the basis of kinetic measurements *in vitro*. *J. Biol. Chem.*, 275(45):34909–34921, 2000. 82

[118] D. Ropers, H. de Jong, M. Page, D. Schneider, and J. Geiselmann. Qualitative simulation of the carbon starvation response in *Escherichia coli*. *Biosystems*, 84(2):124–152, 2006. xiii, 6, 83, 87, 88, 89

[119] I. Sá-Correia, S. Santos, M. Teixeira, T. Cabrito, and N. Mira. Drug:H+ antiporters in chemical stress response in yeast. *Trends Microbiol.*, 17:22–31, 2009. 91, 92

[120] J. Saez-Rodriguez, L. Simeoni, J.A. Lindquist, R. Hemenway, U. Bommhardt, B. Arndt, U.-U. Haus, R. Weismantel, E.D. Gilles, S. Klamt, and B. Schraven. A logical model provides insights into T cell receptor signaling. *PLoS Comput. Biol.*, 3(8):e163, 2007. 3

[121] P.M. Santos, T. Sim oes, and I. Sá-Correia. Insights into yeast adaptive response to the agricultural fungicide mancozeb: A toxicoproteomics approach. *Proteomics*, 3:657–670, 2009. 92

[122] A.U. Shankar. An introduction to assertional reasoning for concurrent systems. *ACM Comput. Surv.*, 25(3):225–262, 1993. 24

[123] X. Shen, J. Collier, D. Dill, L. Shapiro, M. Horowitz, and H.H. Mcadams. Architecture and inherent robustness of a bacterial cell-cycle control system. *Proc. Natl. Acad. Sci. USA*, 105(32):11340–11345, 2008. 4, 21, 65, 77

[124] I. Shmulevich, E.R. Dougherty, S. Kim, and W. Zhang. Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, 2002. 21, 75

[125] P. Starke and S. Roch. Ina - the integrated net analyzer. Humboldt University Berlin, 1999. `http://www2.informatik.hu-berlin.de/~starke/ina.html`. 75

[126] R. Streett. Propositional dynamic logic of looping and converse. *Inform. Control*, 54:121–141, 1982. 29, 35, 49

[127] Z. Szallazi, V. Periwal, and J. Stelling. *System Modeling in Cellular Biology: From Concepts to Nuts and Bolts*. MIT Press, Cambridge, MA, 2006. 56

[128] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacif. J. Math.*, 5:285–309, 1955. 38, 119

[129] M.C. Teixeira, P.J. Dias, T. Sim oes, and I. Sá-Correia. Yeast adaptation to mancozeb involves the up-regulation of FLR1 under the coordinate control of Yap1, Rpn4, Pdr3, and Yrr1. *Biochem. Biophys. Res. Commun.*, 367(2):249–255, 2008. 91, 92, 95

[130] S. Tenreiro, A.R. Fernandes, and I. Sá-Correia. Transcriptional activation of FLR1 gene during *Saccharomyces cerevisiae* adaptation to growth with benomyl: role of Yap1p and Pdr3p. *Biochem. Biophys. Res. Commun.*, 280:216–222, 2001. 92

[131] J.K. Thakur, H. Arthanari, F. Yang, S.J. Pan, X. Fan, J. Breger, D.P. Frueh, K. Gulshan, D.K. Li, E. Mylonakis, K. Struhl, W.S. Moye-Rowley, B.P. Cormack, G. Wagner, and A.M. Naar. A nuclear receptor-like pathway regulating multidrug resistance in fungi. *Nature*, 452:604–609, 2008. 91

[132] R. Thomas. Logical analysis of systems comprising feedback loops. *J. Theor. Biol.*, 73:631–656, 1978. 22

[133] R. Thomas and M. Kaufman. Multistationarity, the basis of cell differentiation and memory: I. Structural conditions of multistationarity and other nontrivial behavior. *Chaos*, 11(1):170–179, 2001. 29

[134] R. Thomas, D. Thieffry, and M. Kaufman. Dynamical behaviour of biological regulatory networks: I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bull. Math. Biol.*, 57(2):247–276, 1995. 21, 39, 75

[135] W. Thomas. Computation tree logic and regular ω-languages. In G. Rozenberg J. W. de Bakker, W-P. de Roever, editor, *Linear time, branching time and partial order in logics and models of concurrency*, volume 354 of *LNCS*, pages 690–713, 1989. 50

[136] P. Wolper. A translation from full branching time temporal logic to one letter propositional dynamic logic with looping, 1982. Unpublished manuscript. 35, 49

[137] P. Wolper. Temporal logic can be more expressive. *Inform. Control*, 56(1/2):72–99, January-February 1983. 50